

Signsoft VisIt – Tutorial 2



The use of VisTube objects

This tutorial will show the usage of VisTube objects for the design of simple and complex 3-dimensional objects.

It is recommended to follow this example step by step. Please create at first a new empty project.

Step 1: Creation of the environment

What is to do?

To display a VisTube object it is necessary to have a view and a camera. Please insert a *VisPerspectiveCamera* and a *VisView* component into your form and change their properties *Align* to *alClient*, and *Camera* to *VisPerspectiveCamera1*. These steps were already described in the first tutorial. To change properties we furthermore need a *TVisAttributes* component, in which we change the property *Lighting* to *true*. Now we put some light into the scene. To move the light source into Z-direction, we set the *Transformation.Translation.Z* to 2. Now our basic components are complete, we only have to render them in the *OnRender* method of *VisView*. To implement this, we call the *Render* method of our *VisAttributes* and *VisLight* components.

If you start the program, you will see only a black screen, because we haven't inserted objects which are to be rendered. To test the correctness of your settings, please insert a *VisSphere* component into your form, and call its *Render* method directly after the *Light*'s method. Now you should see an illuminated sphere in the center of your form. Please remove the sphere object from the form after this text.

The individual steps:

- insert a *TVisPerspectiveCamera* component
- insert a *TVisView* component
- change the *VisView1* property *Align* to *alClient*
- change the *VisView1* property *Camera* to *VisPerspectiveCamera1*
- insert a *TVisAttributes* component
- change the *VisAttributes1* property *Lighting* to *true*
- insert a *TVisLight* component
- change the *VisLight1* property *Transformation.Translation.Z* to 2
- create an handler method for the event *OnRender* of *VisView1*
- in this handler method call the methods *Render* of *VisAttributes1* and of *VisLight1*

Source code:

Delphi:

```
VisAttributes1.Render(aView);  
VisLight1.Render(aView);
```

C++Builder:

```
VisAttributes1->Render(aView);  
VisLight1->Render(aView);
```

Conclusion:

To render a scene it is necessary to create a view and a camera, these should be linked. In most cases it is also useful to create a `VisAttributes` component and a `VisLight` component. Both of them should be rendered in the `OnRender` method of the `VisView`.

Step 2: Use of VisNodes to simplify the scene

What is to do?

Now that you met all prerequisites we may display our objects. In theory, a VisTube component is basically a bent tube with changing diameters. Practically you may model many other things with this concept, as you will see in a couple of moments. In our example we will work on a simple tea-pot. If you look closer, you will see that this task isn't as simple as it seems.

Our tea-pot consists of three VisTube objects, the romp, the spout and the handle. To be able to handle these objects as a single object, we arrange them into a *VisNode* component. Please insert a *TVisNode* component into our form. Now you insert three VisTube components into your form. To link these with the VisNode component you created before, you set their *Node* property to *VisNode1*. To render the VisTubes, you call the *Render* method of *VisNode1*, directly after that of *VisLight1*.

If you start your program, you still won't see anything, because the VisTube objects are empty. You have to append segments to make them visible.

The individual steps:

- insert a *TVisNode* component
- change the *VisNode1* property *Transformation.Rotation.X* to 270 setzen
- change the *VisNode1* property *Transformation.Translation.Z* to -0.5 setzen
- insert three *TVisTube* components
- change the *Node* properties of all three *TVisTube* components to *VisNode1*
- in *VisView1*'s *OnRender* method call *Render* of *VisNode1*

Source code:

Delphi:

```
VisAttributes1.Render(aView);  
VisLight1.Render(aView);  
VisNode1.Render(aView);
```

C++Builder:

```
VisAttributes1->Render(aView);  
VisLight1->Render(aView);  
VisNode1->Render(aView);
```

Conclusion:

To arrange objects it is advisable to use *VisNode* components. Its *Node* property

specifies to which *VisNode* a component belongs. If a *VisNode* is rendered, all objects, which are contained in this node, are rendered, too. By changing the *Transformation* property of a *VisNode*, all contained elements are moved.

Step 3: Introduction to the use of a VisTube

What is to do?



Now we can start the construction of our tea-pot. First we build the romp, basically a straight piece of tube with changing diameters. To create such a tube, we have to append some segments to our still empty *VisTube1*. You may add a straight segment or a bent segment. For our romp we need only straight segments, because only the diameter changes. Later we need bent segments to model our handle and the spout.

To add a straight piece we call the *VisTube* method *AddLine*. Parameters of this method are the destination of the segment, and the target diameter

of the tube. The start diameter is defined by the previous segment, the first diameter is specified in the *VisTube* component itself. How do you start a new object? It is reasonable to use a template, from which the needed segments are "read off" directly. Our tea pot should have the shape as shown in the illustration. The tea-pot is created only once, preferably in the *OnCreate* handler of our form. We start at the base of our tea-pot. The tube shall have a diameter of *0.4*. To do this, change the *VisTube1* property *StartSegment.Radius* to *0.2*. Next we append the first tube segments, At first the radius grows, till the base of our handle. Our new radius is *0.35*, the segment length is *0.36*. The values come from our template, our scale is chosen to give best display. To create this segment, we call the method *AddLine*, with the following parameters:

```
AddLine(0, 0, 0.36, 0.35);
```

This instruction causes a new segment with *0.36* units in Z-direction (up) and unchanged X- and Y-coordinates. Fourth parameter specifies the new radius. Now two more segments are added. First the radius is reduced, until the most narrow spot. Second, the radius grows to the opening on top of the pot. The needed source code is shown below. If you start your program, you will see a rather rough version of our pot. The radii and the segment lengths are basically correct, but the transition is somewhat un-smooth. We will fix that in the next chapter.

The individual steps:

- create a handler method for the *OnCreate* event of our form
- insert the given source code to create the first segments.

Source code:**Delphi:**

```
VisTubel.StartSegment.Radius:=0.2;  
  
VisTubel.AddLine(0,0,0.36, 0.35);  
VisTubel.AddLine(0,0,0.25, 0.19);  
VisTubel.AddLine(0,0,0.1, 0.225);
```

C++Builder:

```
VisTubel->StartSegment->Radius=0.2;  
  
VisTubel->AddLine(0,0,0.36, 0.35);  
VisTubel->AddLine(0,0,0.25, 0.19);  
VisTubel->AddLine(0,0,0.1, 0.225);
```

Conclusion:

You may add a straight tube segment to a *VisTube* by calling the *AddLine* method. The first three parameters specify the position of the end of the segment, in relation to the coordinates of the segment start. The fourth parameter specifies the radius at the end of the tube segment. The *start radius* of the *VisTube* is specified in the *StartSegment.Radius* property.

Step 4: Transitions between segments of varying width

What is to do?

In spite of our tea-pot having the correct values for the needed segments, the pot displayed quite differs from our template. Instead of some nice, curvy tea-pot we have got some ugly sharp-edg. container. The reason is simple: with our calls we only specified the radii of the curve segments, we didn't specify, how the transition between the segments should be rendered. To add this, we introduce two new parameters to our *AddLine* method. The first is the amount of sub-segments or stacks, which our segment will consist of. Perhaps you already know the *stacks* property from other 3D objects. A higher *stack* value yields more details, but also requires a high effort for calculating. You may select this value as a fifth parameter on the *AddLine* method. The default value of this parameter is *10*. Another optional value defines the transition between the starting radius and the end radius of your curve segment. The parameter *RadChange* can have the constant values as shown in the illustration. Apart from that the illustration shows how the respective value affects the enlargement and reduction of the radius. The default value of this parameter is *rcLinear*. This is the reason for the rather sharp-edged appearance of our tea-pot. You can solve this problem by adding a value for *stacks* and one for the transition to each call to *AddLine*. The needed values are taken from our template. The first segment will have a *rcEndSmooth* value, the second has *rcSmooth*, the third has *rcStartSmooth*. After you have started the program, you will see the finished first part of our tea-pot. To see the influence of the possible values of *RadChance*, you can play around with them. As you will see, with only three calls you can get most different forms.

Source code:

Delphi:

```
VisTubel.AddLine(0,0,0.36, 0.35, 10, rcEndSmooth);  
VisTubel.AddLine(0,0,0.25, 0.19, 10, rcSmooth);  
VisTubel.AddLine(0,0,0.1, 0.225, 5, rcStartSmooth);
```

C++Builder:

```
VisTubel->AddLine(0,0,0.36, 0.35, 10, rcEndSmooth);  
VisTubel->AddLine(0,0,0.25, 0.19, 10, rcSmooth);  
VisTubel->AddLine(0,0,0.1, 0.225, 5, rcStartSmooth);
```

Conclusion:

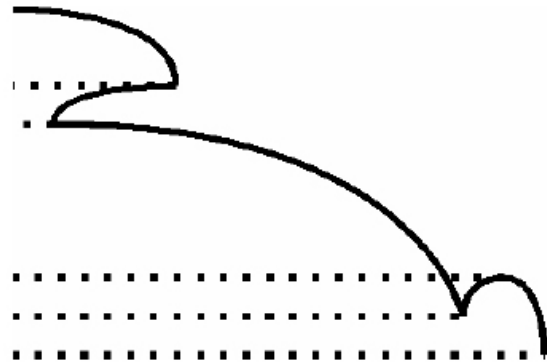
In most cases it is necessary to not only specify the end radius of a segment, but also to specify, how the transition between both end values shall be. This can be done by the parameter *RadChange*, which is given as sixth parameter to

AddLine. You have to consider if the radius reduces or grows. To specify the level of details along a tube segment, you may specify the *stacks* parameter.

Step 5: Construction of more detailed shapes

What is to do?

In the current state of the creation of our tea-pot there are some parts missing. We need a top border with a lid. Pot and lid are made of one tube. If we want to remove the lid, both should be modeled as two separate VisTubes. A cross section of the upper border of our tea-pot shows the needed segments. A hatched line shows the needed end radius of a segment. The last segment finishes with a radius of 0, so we get a closed lid.



As you can see, we need 5 more segments. If you create these segments you will see an unwanted effect in the lighting of our tea-pot. On the sharp transition below the handle the curve changes rapidly its direction. This causes also a sharp difference in the illumination. OpenGL creates a smooth transition from dark to light. Normally this is useful, but not in this case. To circumvent this effect, we create an additional very short segment. The created smooth transition is barely visible, really not longer recognizable. Another way to create such an effect would be the creation of separate VisTube objects. The values for the needed calls are shown below. Beside the lid our tea-pot is missing a bottom. This happened because we started with the border of the tea-pot. To finish our pot on the bottom, we change our start radius to 0 and insert another line segment which only increases the radius to our old start radius. The segment has no other coordinate moves.

Source code:

Delphi:

```

VisTubel.StartSegment.Radius:=0;
VisTubel.AddLine(0,0,0, 0.2, 1);

VisTubel.AddLine(0,0,0.36,0.350,10,rcEndSmooth);
VisTubel.AddLine(0,0,0.25,0.190,10,rcSmooth);
VisTubel.AddLine(0,0,0.10,0.225, 5,rcStartSmooth);

VisTubel.AddLine(0,0, 0.010,0.215,5,rcStartSmooth);
VisTubel.AddLine(0,0,-0.005,0.210,5,rcEndSmooth);

VisTubel.AddLine(0,0,0.08, 0.03,10,rcStartSmooth);
VisTubel.AddLine(0,0,0, 0.029,1);

```

```
VisTubel.AddLine(0,0,0.02, 0.06,10,rcSmooth);  
VisTubel.AddLine(0,0,0.03, 0 ,10,rcStartSmooth);
```

C++Builder:

```
VisTubel->StartSegment->Radius=0;  
VisTubel->AddLine(0,0,0, 0.2, 1);  
  
VisTubel->AddLine(0,0,0.36,0.350,10,rcEndSmooth);  
VisTubel->AddLine(0,0,0.25,0.190,10,rcSmooth);  
VisTubel->AddLine(0,0,0.10,0.225, 5,rcStartSmooth);  
  
VisTubel->AddLine(0,0, 0.010,0.215,5,rcStartSmooth);  
VisTubel->AddLine(0,0,-0.005,0.210,5,rcEndSmooth);  
  
VisTubel->AddLine(0,0,0.08, 0.03,10,rcStartSmooth);  
VisTubel->AddLine(0,0,0, 0.029,1);  
VisTubel->AddLine(0,0,0.02, 0.06,10,rcSmooth);  
VisTubel->AddLine(0,0,0.03, 0 ,10,rcStartSmooth);
```

Conclusion:

VisTube objects allow the creation of realistic details. The advantage of this approach is that even complex shapes can be created without resorting to several separate pieces, but can be handled as one object. Unwanted lighting effects can be minimized by additionally inserting very small tube segments.

Step 6: Creation of bent tubes

What is to do?

Next we model the handle of our tea-pot. As you can see, it can be created with two simple calls. Because the handle starts on the side of the rump, we have to put the `VisTube2` object into the correct position. Furthermore the base area of our tube starts in the `XY` -plane, i.e. it starts, like the bottom of our teapot. The tube shape also depends on the adjustment of its output surface. Imagine that the bottom of the tea-pot is rotated in any direction. All segments would end at the same positions, but the shape of the element would be rather different. If we create our handle we have to consider that initially it is lying rapidly at the bottom. Later it will be transformed into its correct position.

To create our handle we need a new method, which will create a bent tube. This function is called `AddBend` and works similar as you know from the `AddLine` method. The only difference is that the transition between start and end radius is shaped as an arc instead of a line. You may also specify the number of stacks, and the transition of the radius along the segment. You may see the needed calls below. The relative coordinates are taken from the template, but consider that the handle is rotated 90 degrees only after it was created.

If you follow the next steps, you already created the complete handle. It rises to the interior of the tea-pot, but this doesn't matter, because you can not see the inner part. If you want to achieve this, both elements, the rump and the handle, have to be adapted accordingly.

The individual steps:

- change the property `Transformation.Rotation.Y` of `VisTube2` to 90
- change the property `Transformation.Translation.X` of `VisTube2` to 0.2
- change the property `Transformation.Translation.Z` of `VisTube2` to 0.25
- change the property `Transformation.Order` of `VisTube2` to `toTraRotSca`
- insert the given calls into the form's `OnCreate` method

Source code:

Delphi:

```
VisTube2.StartSegment.Radius:=0.04;  
VisTube2.AddBend(-0.3,0, 0.20, 0.03);  
VisTube2.AddBend( 0 ,0,-0.25, 0.04);
```

C++Builder:

```
VisTube2->StartSegment->Radius=0.04;  
VisTube2->AddBend(-0.3,0, 0.20, 0.03);  
VisTube2->AddBend( 0 ,0,-0.25, 0.04);
```

Conclusion:

You may append bent and straight segments to a VisTube. The needed method is called *AddBend*. Its parameters are the same as for *AddLine*, but arcs/bends will be created. The bend shape is very dependent on the adjustment of its output surface.

Step 7: Creation of more complicated shapes

What is to do?

No all we have left to do is adding the spout. This piece with its curvy shape is slightly more complicated than the handle. But first we have to solve a problem we already met with the handle. This time we use some vector analysis.

The spout would have to be rotated by 90 degrees, because it has to start on the side of the rump, not at its bottom. Beside the possibility of rotating the whole object, we also may only rotate the base surface. This surface is specified by two vectors, which are positioned orthogonally. You may visualize this as some kind of coordinate system which is freely positioned in space. The default setting shows a vector in X-direction, the other in Y-direction. To title the plane all we have to do is to provide the appropriate vectors. These vectors should stay orthogonally, and their product should remain *one*. In our case we define our vectors by setting the property *StartSegment.VecX* of *VisTube3* to $(0,0,1)$.

Naturally the base of the spout has to be translated to the correct position. After that we can start appending our bends. We only need 4 calls to *AddBend*, the parameter values are shown below. In real life it is rather complicated to find the needed values. This is because the shape of the spout is specified rather exactly. To model such elements, it is preferable to get some base values from our template. These values are used to create the needed segments. After that the segments can be adjusted accordingly. While segmenting the needed shape it is important to remember, that only regular line and bend segments can be modeled. Other shapes can be generated by creating several sub-segments, until the needed result is reached. If you run the program, you see a nearly perfect tea-pot, only the spout still looks like that of a watering can. In the next chapter we will take care of that and some other subtleties.

The individual steps:

- change the property *Transformation.Translation.X* of *VisTube3* to -0.3
- change the property *Transformation.Translation.Z* of *VisTube3* to 0.25
- insert the given calls into the form's *OnCreate* method

Source code:

Delphi:

```
VisTube3.StartSegment.Radius:=0.13;  
VisTube3.StartSegment.VecX.X:=0;  
VisTube3.StartSegment.VecX.Z:=1;  
  
VisTube3.AddBend(-0.10,0,0.10,0.07,10,rcSmooth);  
VisTube3.AddBend(-0.01,0,0.20,0.05,10,rcSmooth);
```

```
VisTube3.AddBend(-0.03,0,0.09,0.04,3,rcSmooth);  
VisTube3.AddBend(-0.03,0,0.03,0.04,3,rcSmooth);
```

C++ Builder:

```
VisTube3->StartSegment->Radius=0.13;  
VisTube3->StartSegment->VecX->X=0;  
VisTube3->StartSegment->VecX->Z=1;  
  
VisTube3->AddBend(-0.10,0,0.10,0.07,10,rcSmooth);  
VisTube3->AddBend(-0.01,0,0.20,0.05,10,rcSmooth);  
  
VisTube3->AddBend(-0.03,0,0.09,0.04,3,rcSmooth);  
VisTube3->AddBend(-0.03,0,0.03,0.04,3,rcSmooth);
```

Conclusion:

Beside the starting radius, also the adjustment of its base surface can be specified by adapting the property *StartSegment* of *VisTube*. To do this, you have to change the vectors, which describe the base surface. The shape of a tube and the adjustment of its end surface are dependent on the given coordinates and adjustment of its start surface. If you model more complicated shapes, it can become feasible to test different micro adjustments, to yield optimal results. The radius of a segment and its shape have no influence to the appearance of a tube.

Step 8: Extended settings and optimizations

What is to do?

Our tea-pot is almost ready. Only the spout needs some enhancements. The problem is neither its radius nor its course, but its end angle. Please compare with our template. The angle at the end of the tube is chosen automatically to get a regular arc.

There is also the possibility to get a different angle. This effect can't be reached by giving parameters to the *AddBend* or *AddLine* methods. On the other hand, we have the possibility to change settings on our tube segments, even after we appended them to the tube. We can use the property *Sections*, which provides access to the individual segments. To access a segment, we specify an index. The first segment has the index 0. The last segment has an index which is one less than the number of segments. The number of segments can be accessed by the property *SectionCount*. To change the angle at the end of the spout we change the properties *EndAngleX* and *EndAngleY* of the appropriate tube segment. In our case we only need to change the second angle. The first property allows to change the opening to and from the direction of our camera. The new value can be selected by testing. In theory it should be possible to calculate the exact value, but this approach is rather complicated, because the surface, where the segment starts, has to be taken into account. The needed source code is shown below.

Now our tea-pot is complete. You may move the three used VisTube objects like one by using the VisNode1 object. You may also attach textures. If you change the stacks property you can increase the level of detail. Recommended values are 30 at the VisTube1 object, VisTube2 and VisTube3 could have values of 20. The methods *AddLine* and *AddBend* return the references to the added segments, these references can be used to change the segments right after their creation.

Source code:

Delphi:

```
VisTube3.Sections[VisTube3.SectionCount-1].EndAngleY:= -40;
```

C++Builder:

```
VisTube3->Sections[VisTube3->SectionCount-1]->EndAngleY= -40;
```

Conclusion:

In special cases it may be feasible to change the angle at the end of a tube segment. This is possible by accessing an already created segment with the

Sections property. After having obtained a reference to an segment, it is possible to change its `EndAngleX` and `EndAngleY` properties. The complete program, extended by a simple function for the rotation of the tea-pot, can be found in the Signsoft VisIt example programs. There is also another example about the creation of complex objects with the help of VisTube component.