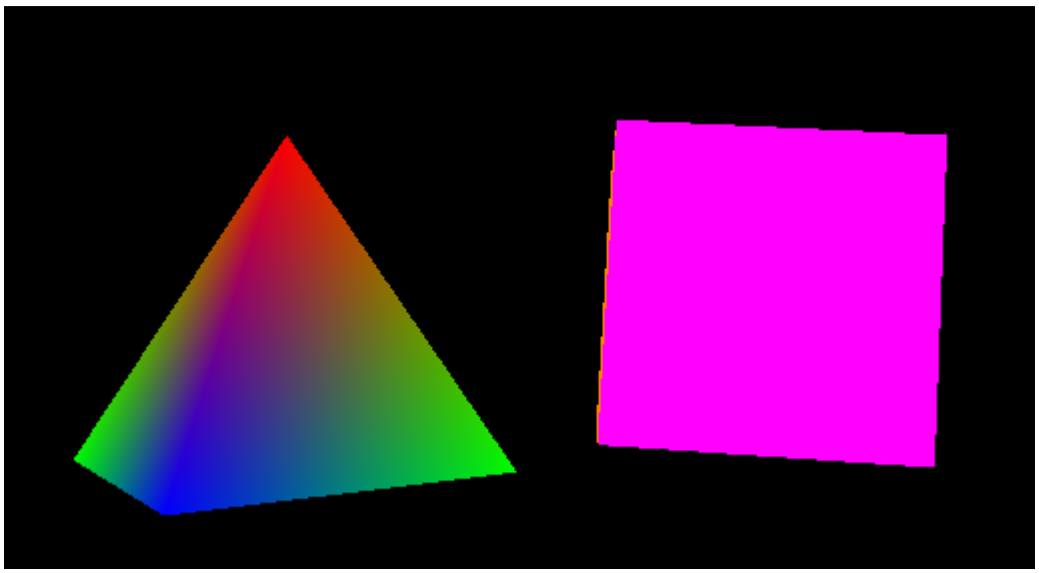
Sharp GL – Simple Drawing

Contents

Introduction

This sample draws a rotating pyramid and a rotating cube using OpenGLControl in SharpGL library.



This sample uses SharpGL compiled to Microsoft .Net Framework 4.0. Limnor Studio VS and Limnor Studio 5 do not support .Net Framework 4.0. Please use Limnor Studio 5.6 which supports .Net Framework 4.0.

The files for this sample project can be downloaded from:

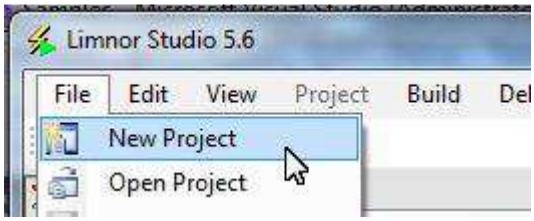
http://www.limnor.com/studio/SharpGL_SimpleDrawing.zip

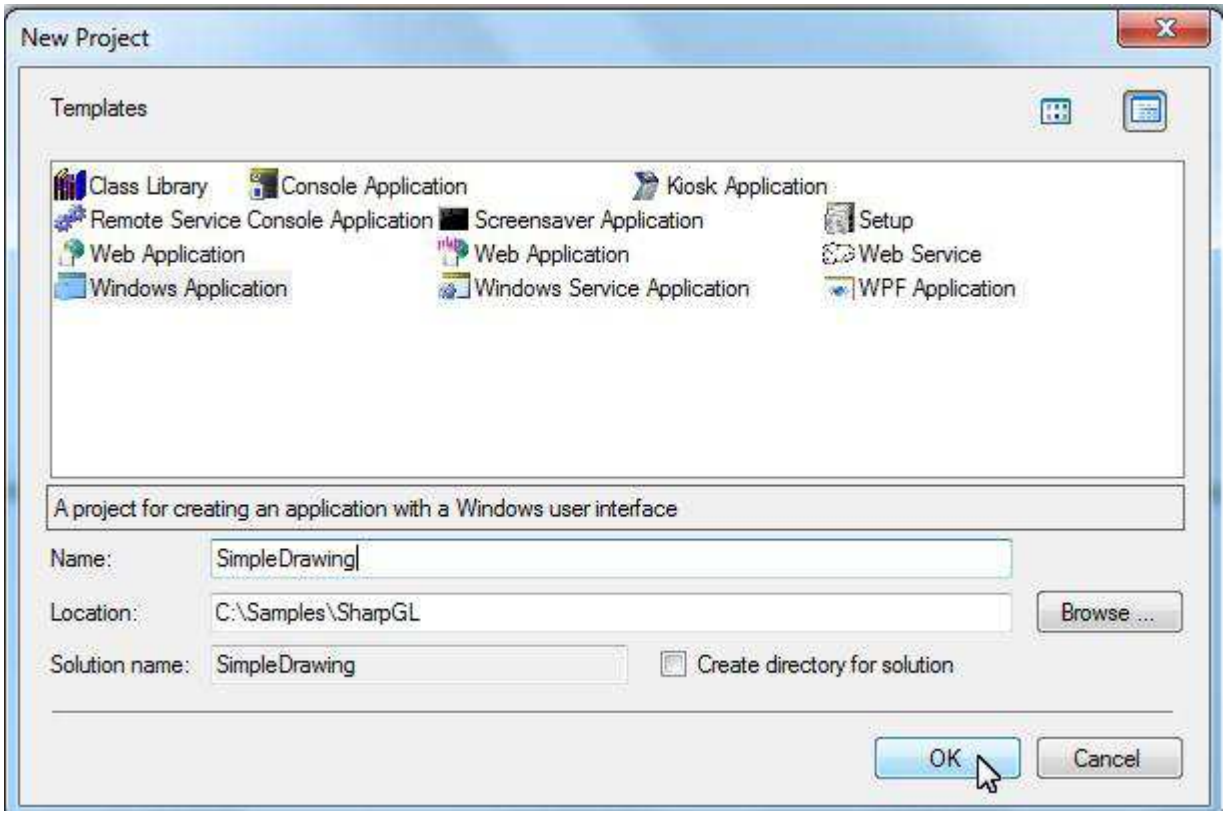
A set of SharpGL DLL files is included in the above zip file. When you open the sample project in Limnor Studio and it asks you to fix the SharpGL DLL locations you may point them to the set of DLL files.

SharpGL is available at http://sharpgl.codeplex.com/.

Load SharpGL Control

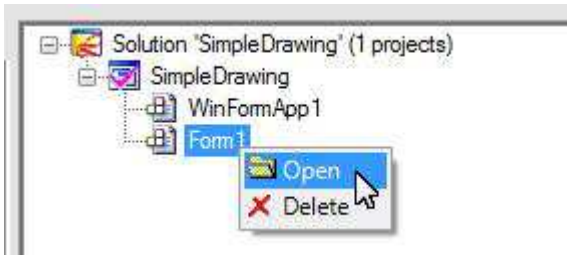
Create a Form application project with Limnor Studio 5.6:



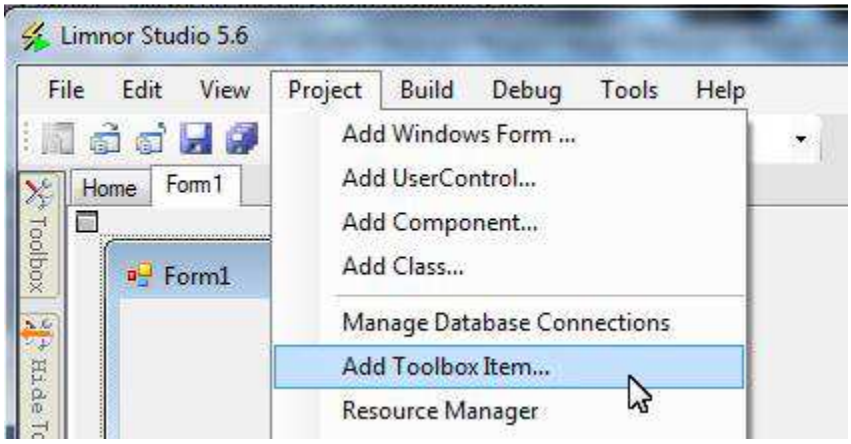


Add OpenGLControl control to the Toolbox:

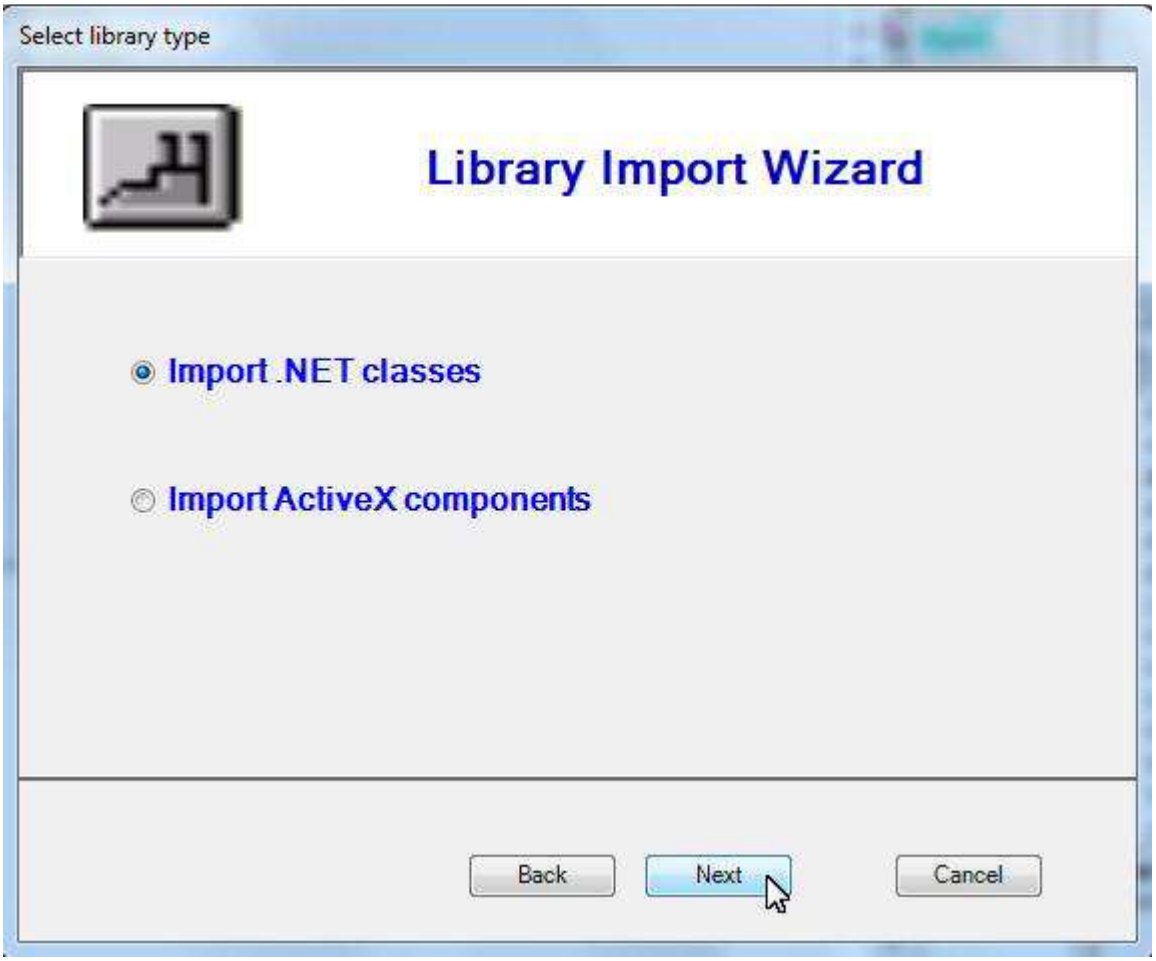
Open the form into the designer:



Select “Add Toolbox Item …”:

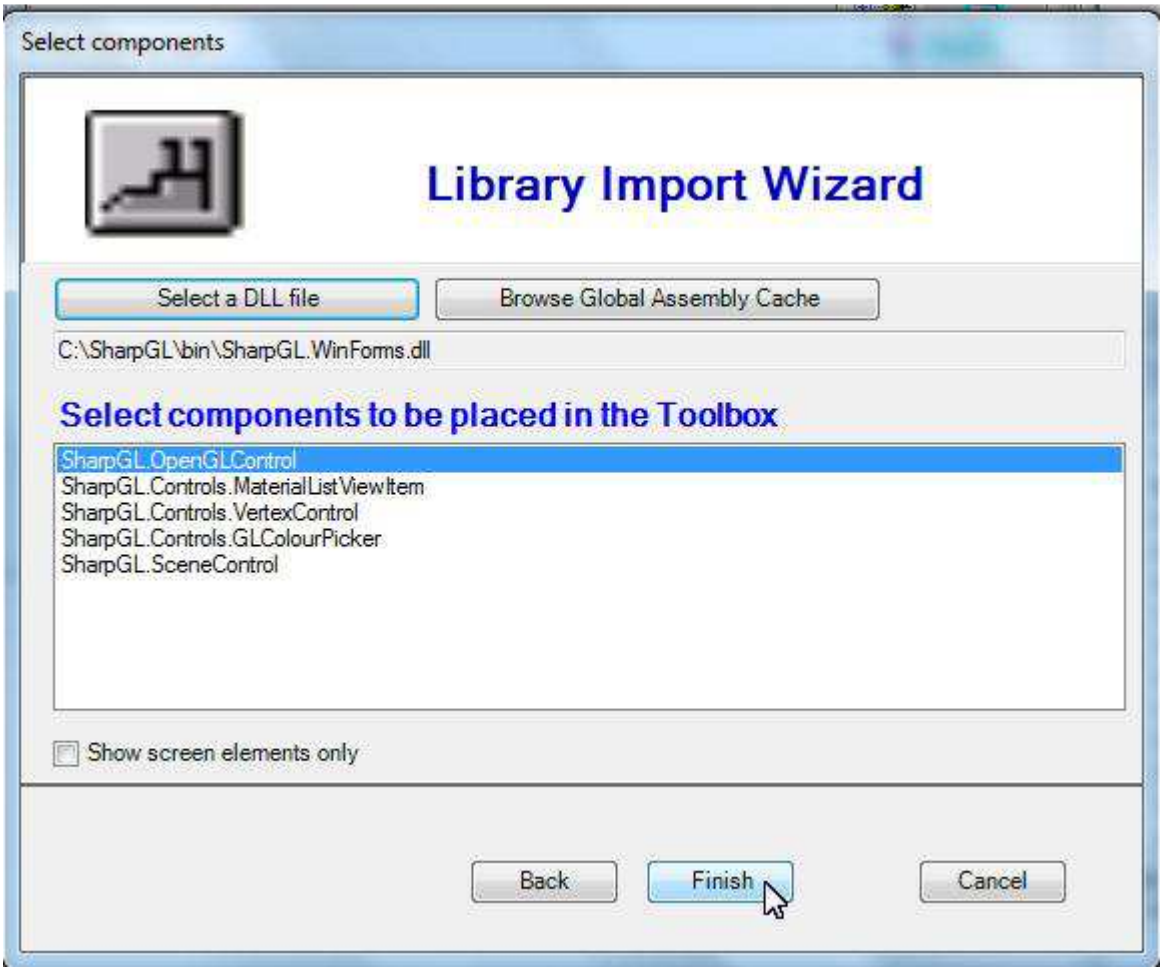

Limnor Studio 5.6
File
Edit
View
Project
Build
Debug
Tools
Help
Add Windows Form ...
Add UserControl...
Add Component...
Add Class...
Manage Database Connections
Add Toolbox Item...
Resource Manager
Home
Form1
Form1
Toolbox


Library Import
Library Importer
This tool imports ActiveX components or Microsoft .NET classes into your projects. The imported components will appear in the Toolbox under the tabs named after your projects.
To import an ActiveX component some new DLL files will be created in your project folder to wrap the ActiveX component to be imported in .Net classes.
Start
Finish

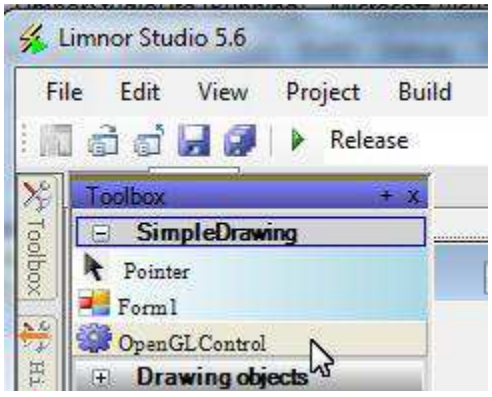


Select file SharpGL.WinForms.dll from the SharpGL file list:

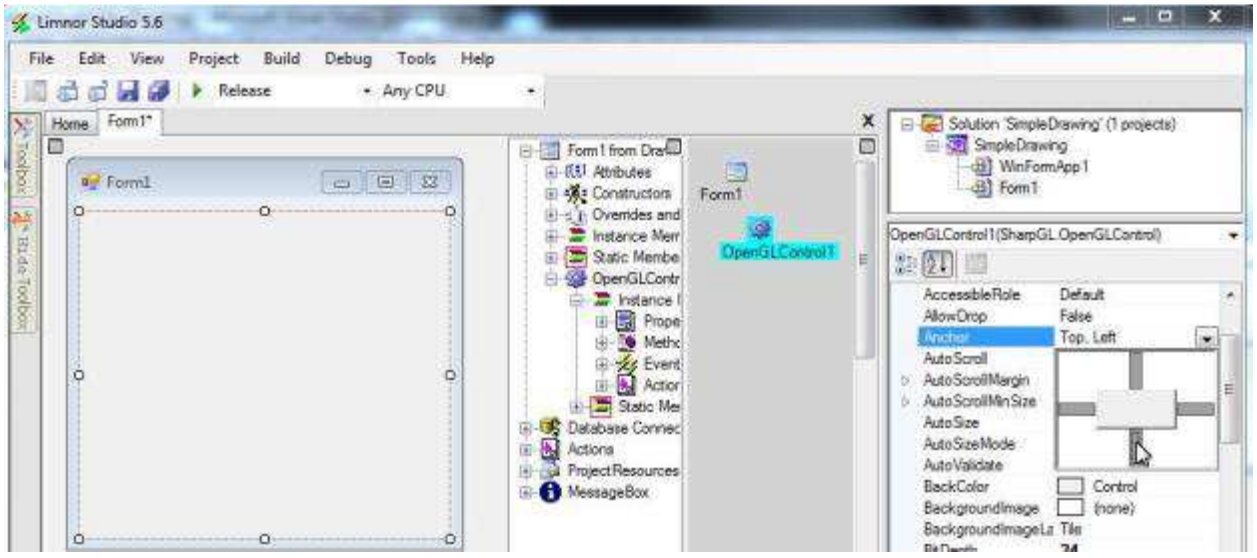
Select OpenGLControl and click Finish:



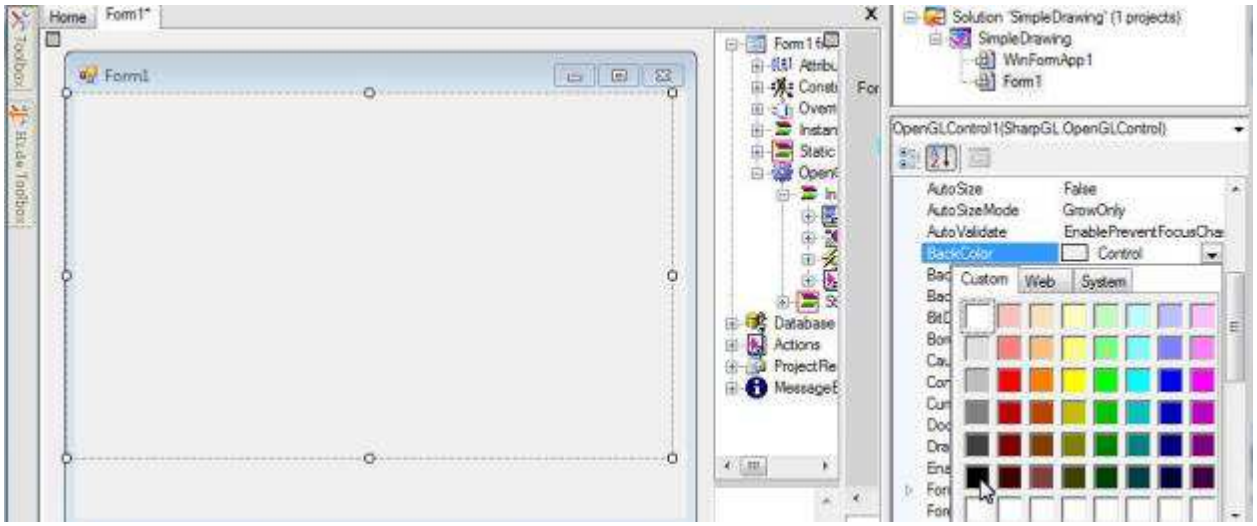
OpenGLControl appears in the Toolbox:

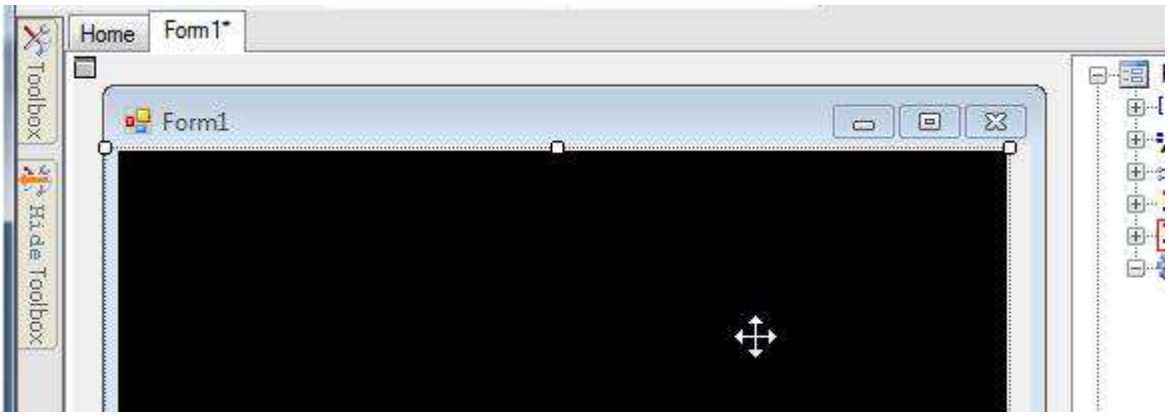


Drop it to the form. Set its Anchor property to anchor on all sides:

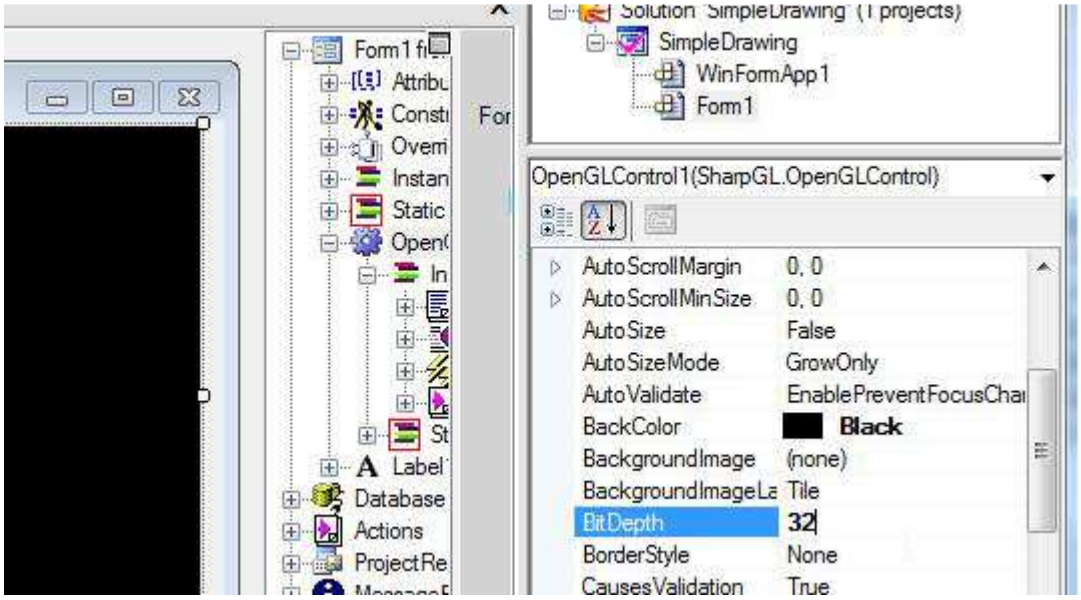


Set the control background color to black:

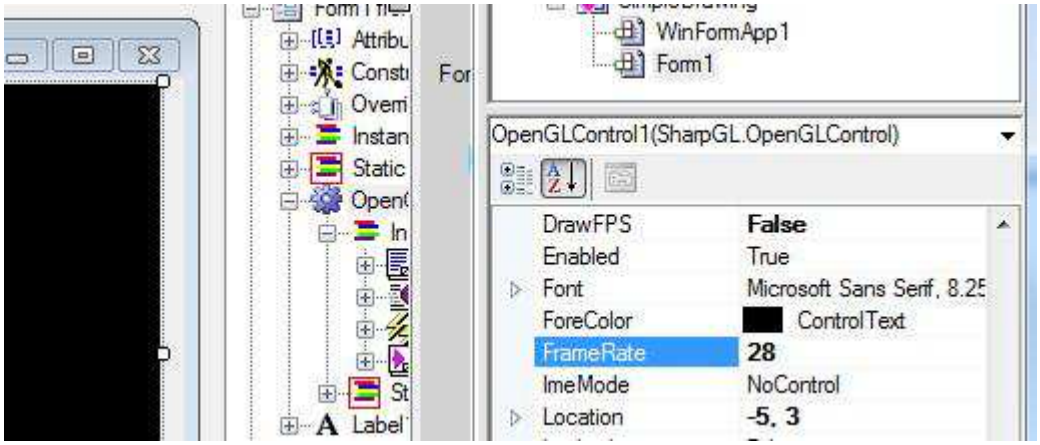




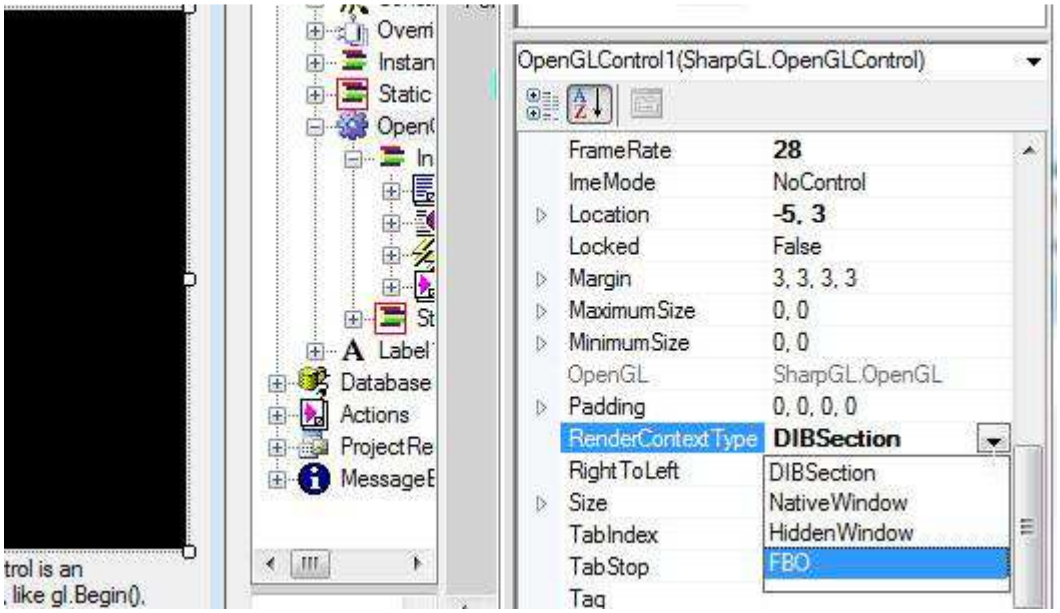
Set BitDepth property to 32:



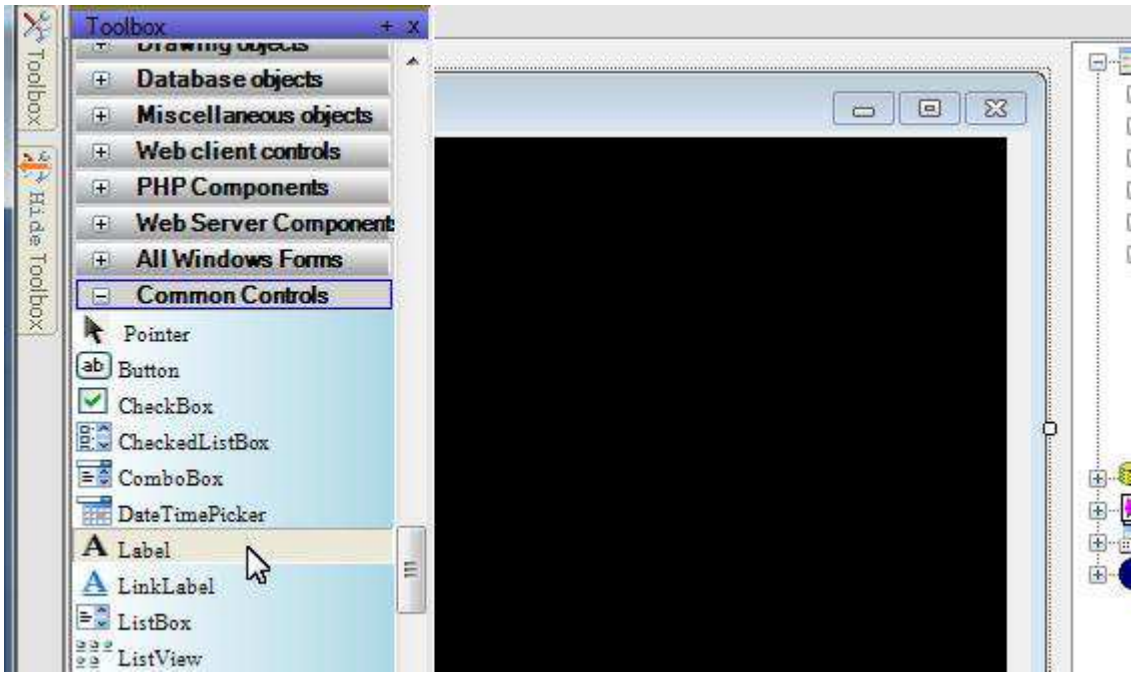
Set FrameRate property to 28:



Set RenderContextType to FBO:



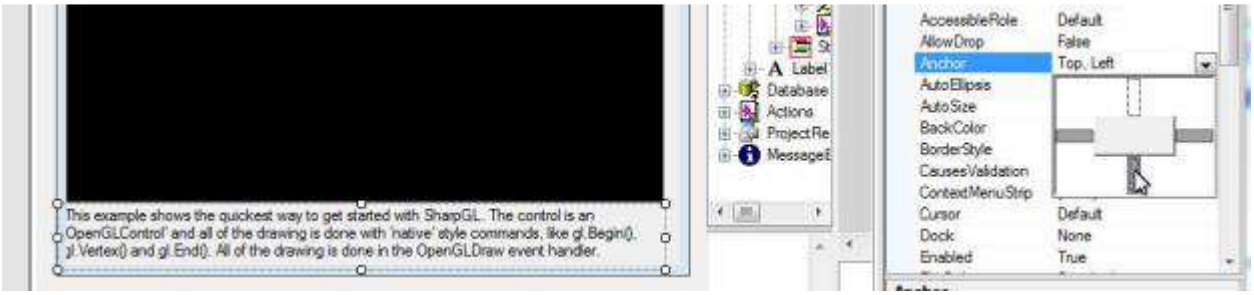
Add a label for description:



Set the Text of the label to following text:

This example shows the quickest way to get started with SharpGL. The control is an 'OpenGLControl' and all of the drawing is done with 'native' style commands, like Begin(),Vertex() and End(). All of the drawing is done in the OpenGLDraw event handler. 'FrameRate' has been set to about 28 Hz.

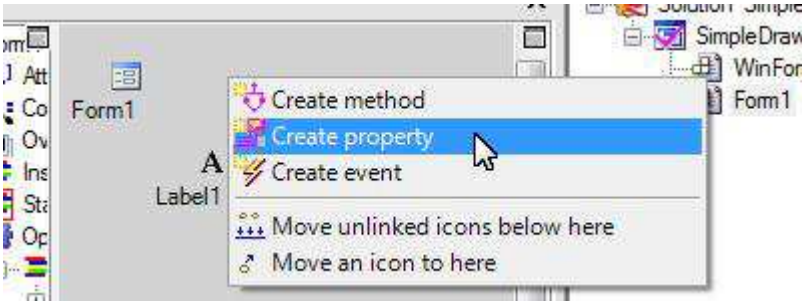
Set Anchor property of the label to {left, right, bottom}:



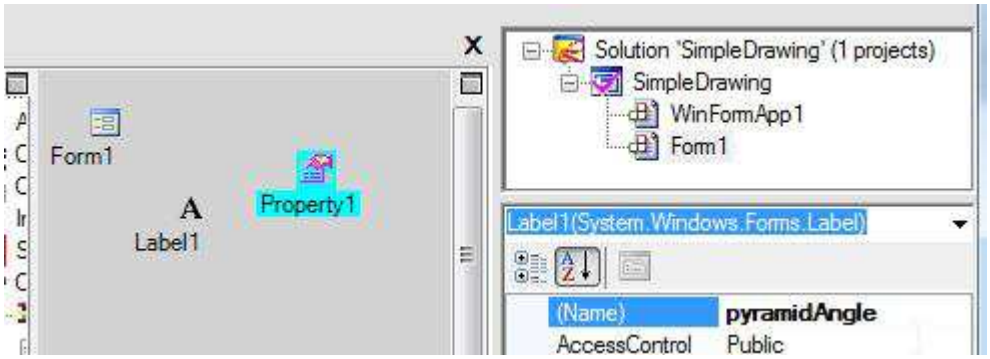
Make drawings

Rotation angles

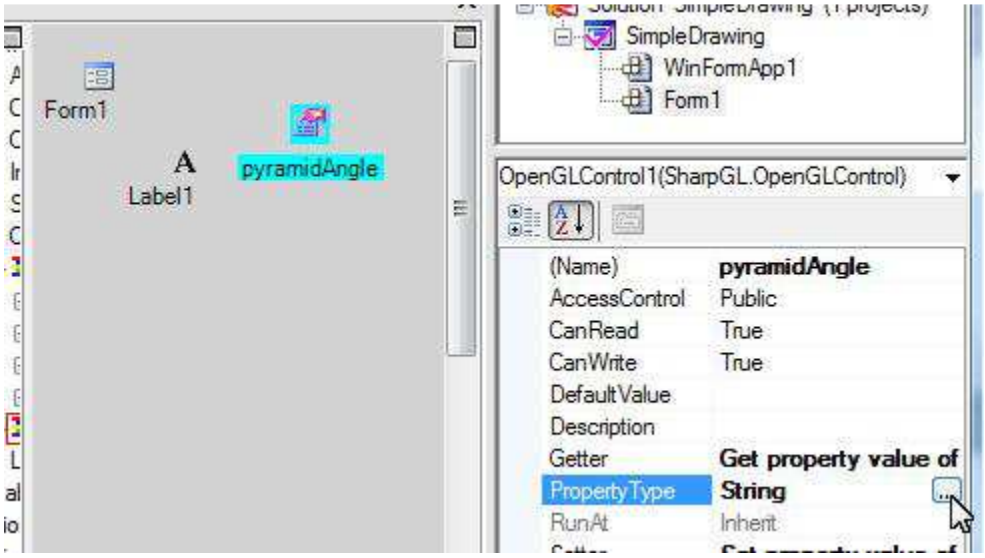
We create two properties for pyramid rotation angle and cube rotation angle respectively.

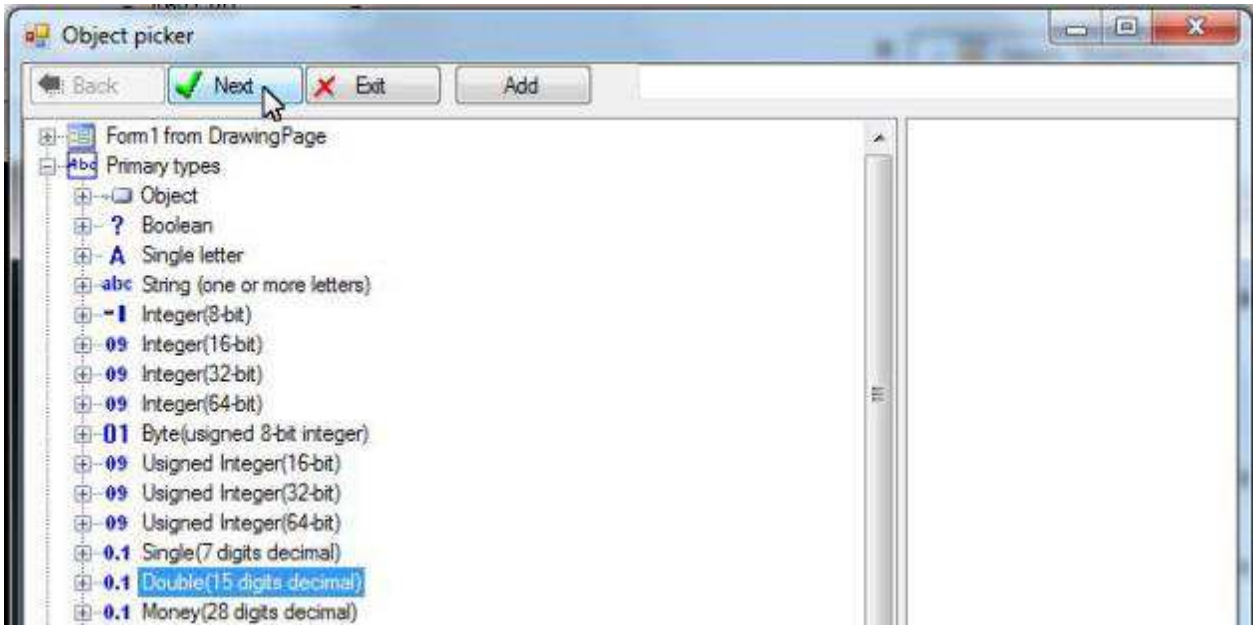


Name it pyramidAngle:

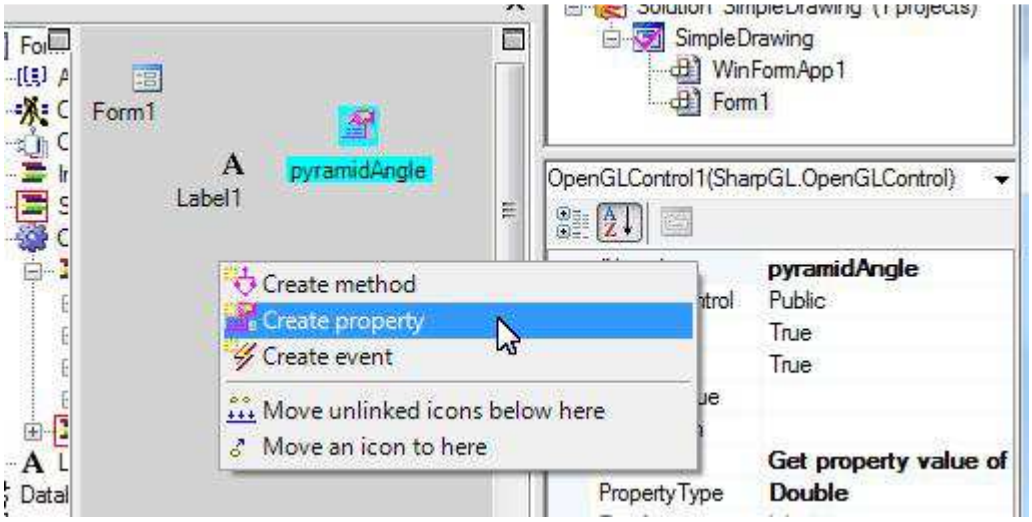


Change its data type to be decimal:

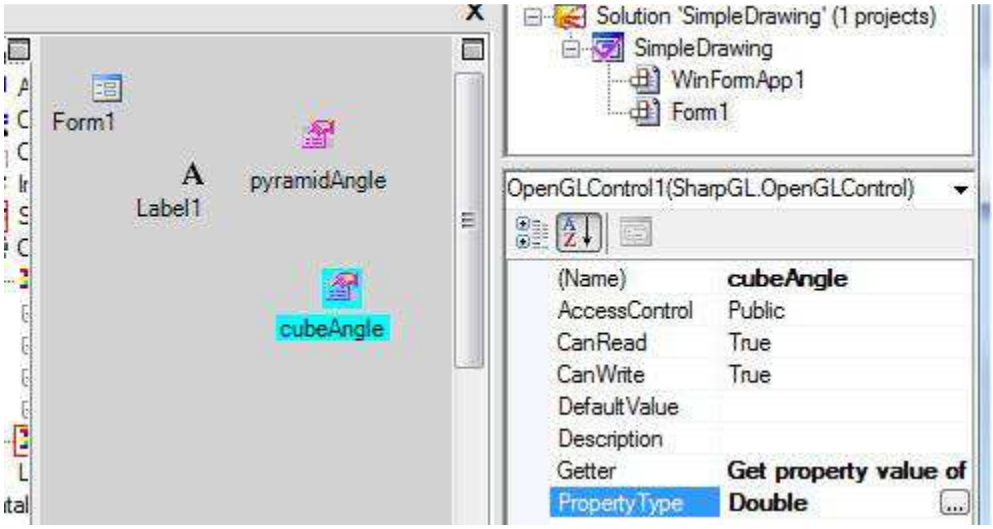




Add a cubeAngle property:

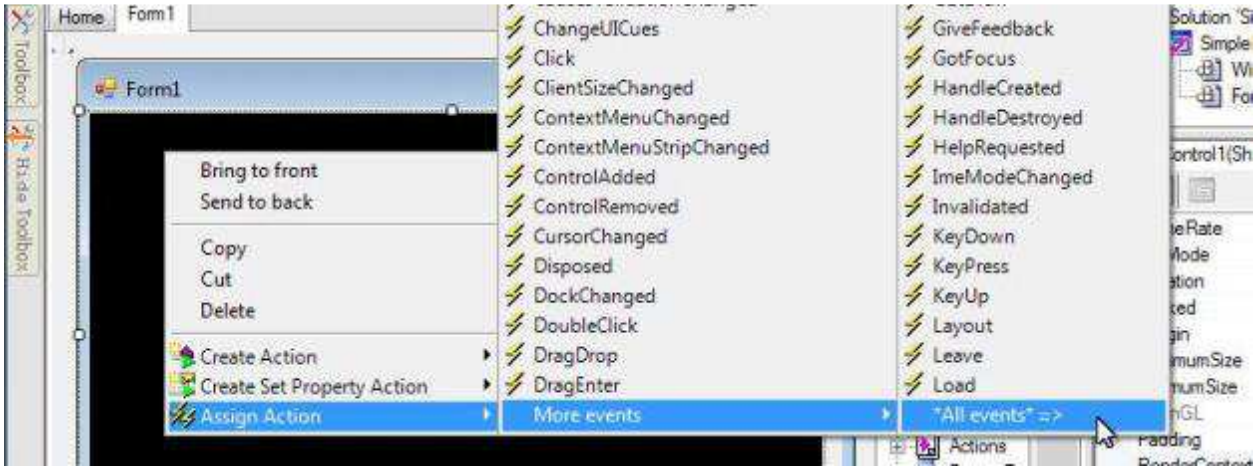


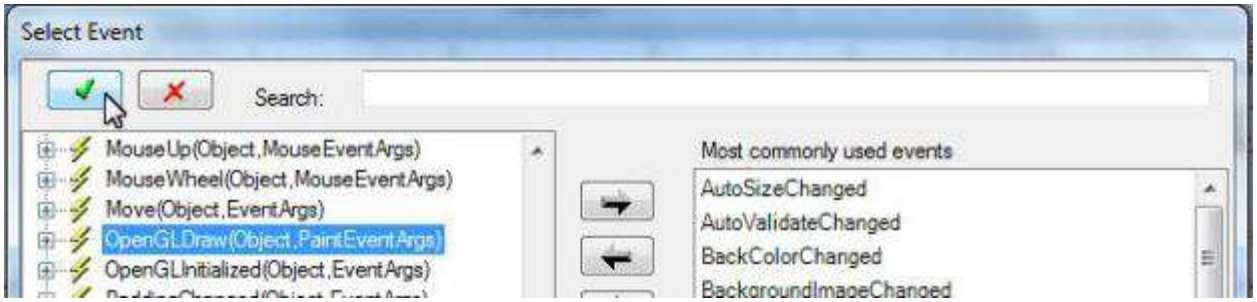
Make it a decimal:

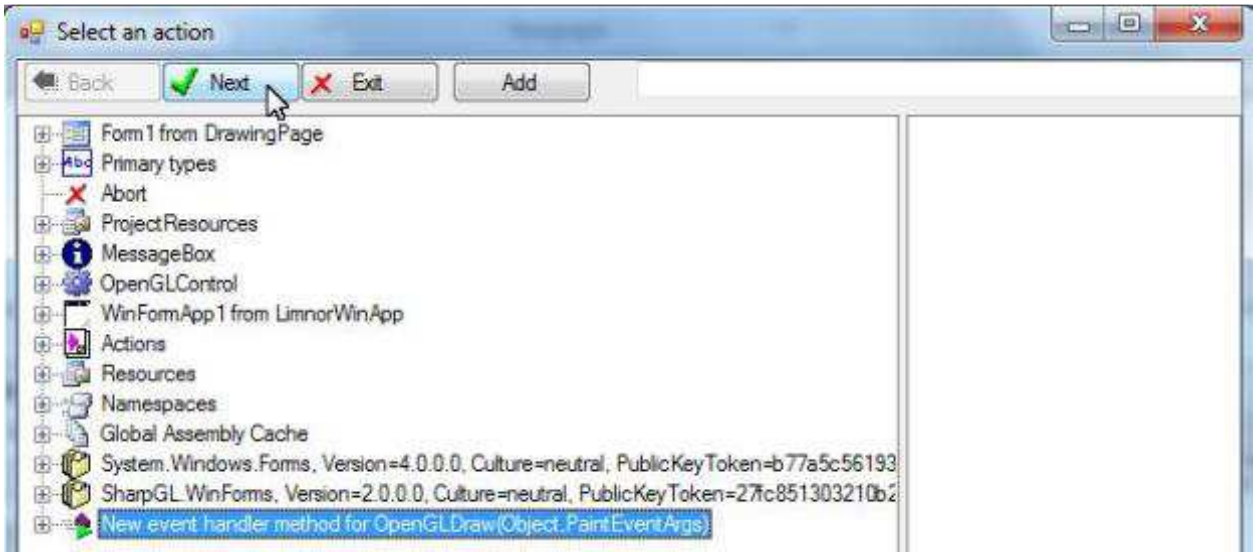


Handle OpenGLDraw event

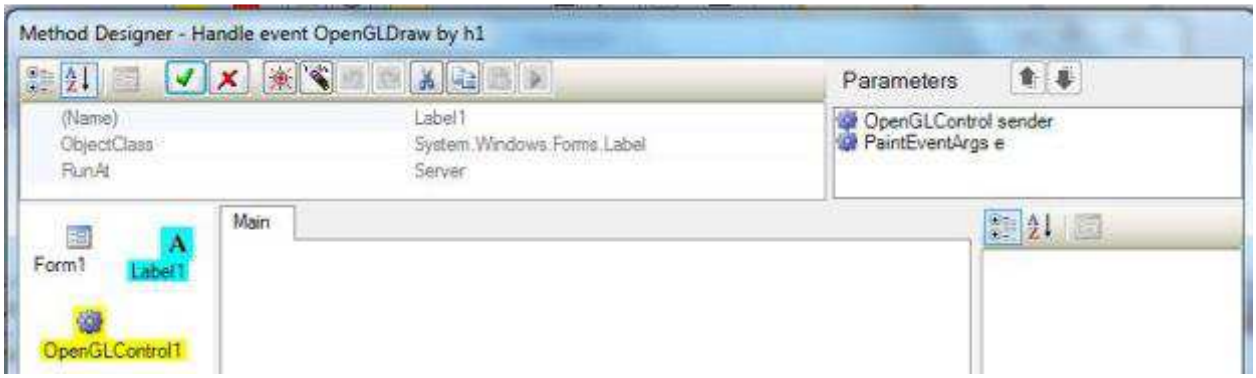
The drawing actions should be used when handling OpenGLDraw event. One event is for drawing one frame. Let's create an event handler method for event OpenGLDraw:





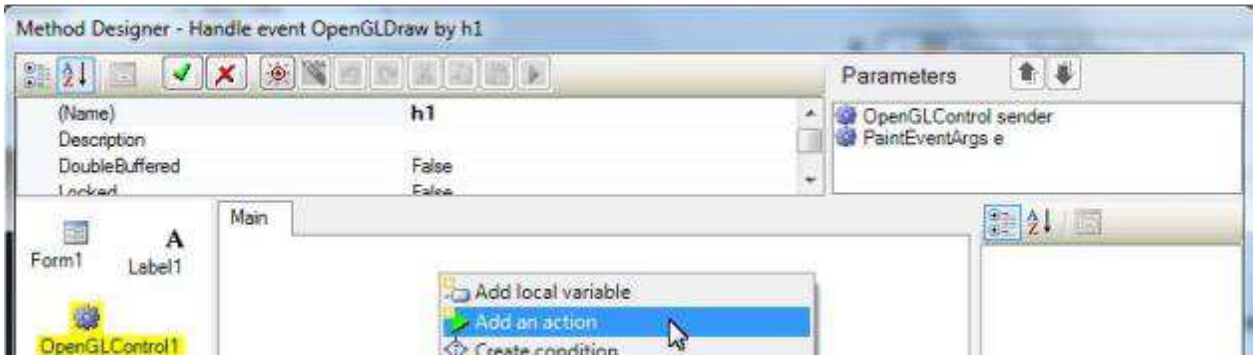


A method editor appears for adding actions to the event:

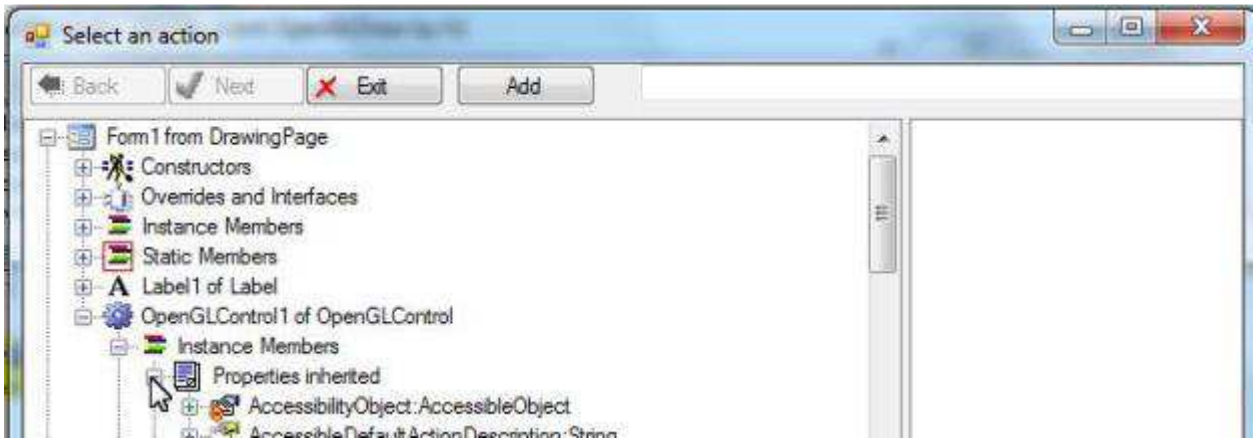


Prepare for drawing – clear buffers

Create an action to clear screen and depth buffer. Right-click to add new action:

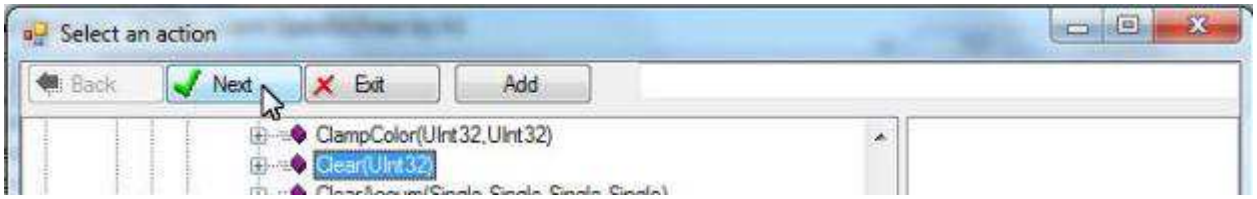


All OpenGL actions are created by OpenGL property of the OpenGLControl. Locate OpenGL property:

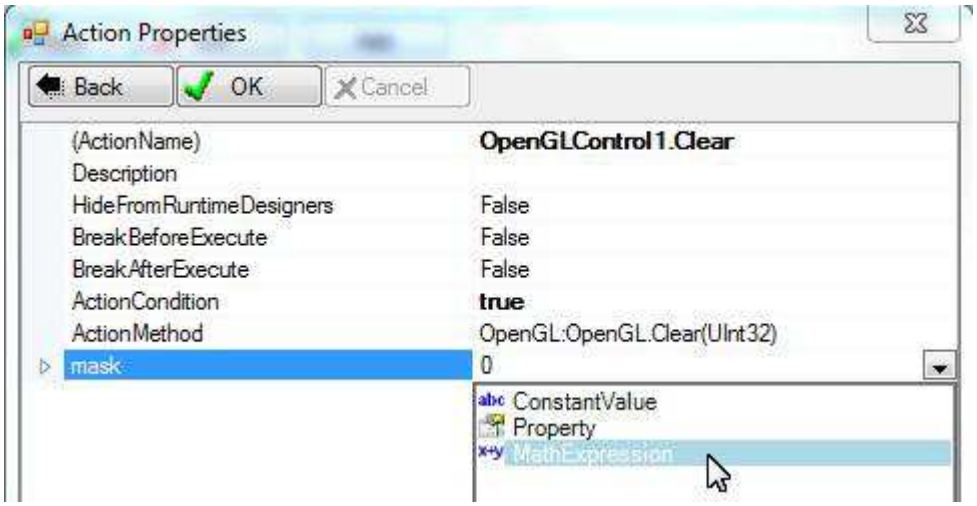


Use the methods listed under OpenGL property to create OpenGL actions:

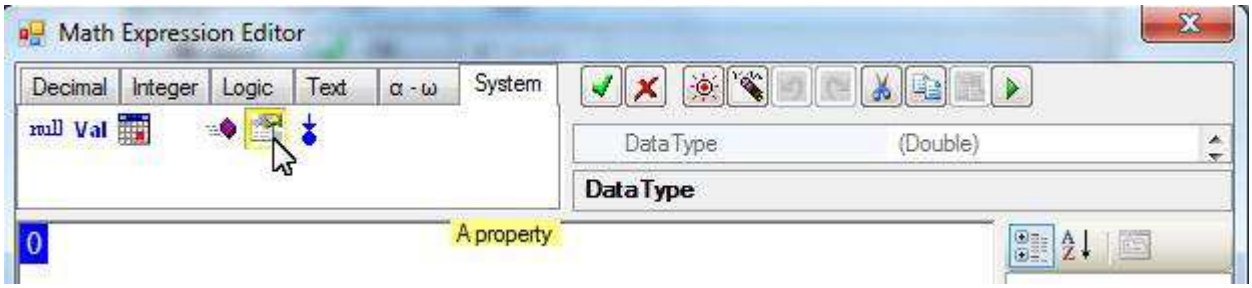
The method to clear screen and depth buffers is Clear:



Use Math Expression to form the mask parameter:



Use Property icon to select a mask:



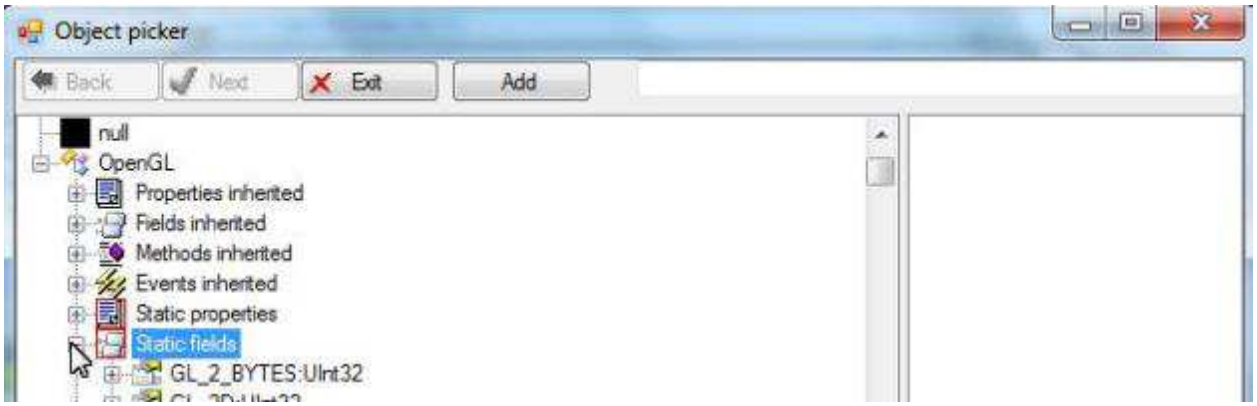
The mask definitions are in core SharpGL library. Click Add to load it:

Click the file button to locate the DLL file:

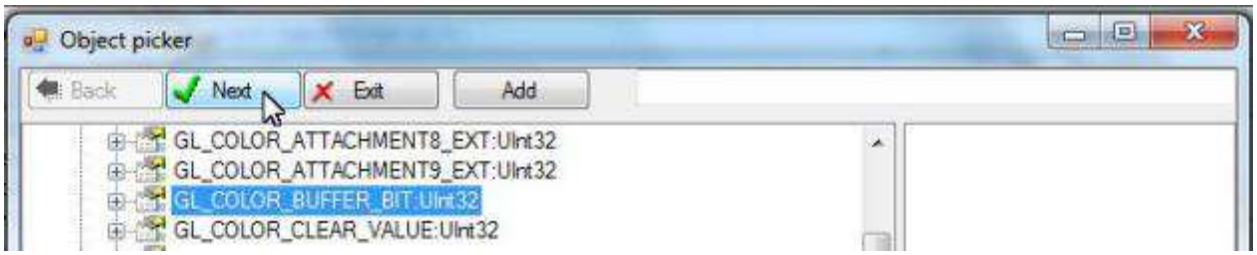
Select SharpGL.DLL:

SharpGL appears. We may find OpenGL under it:

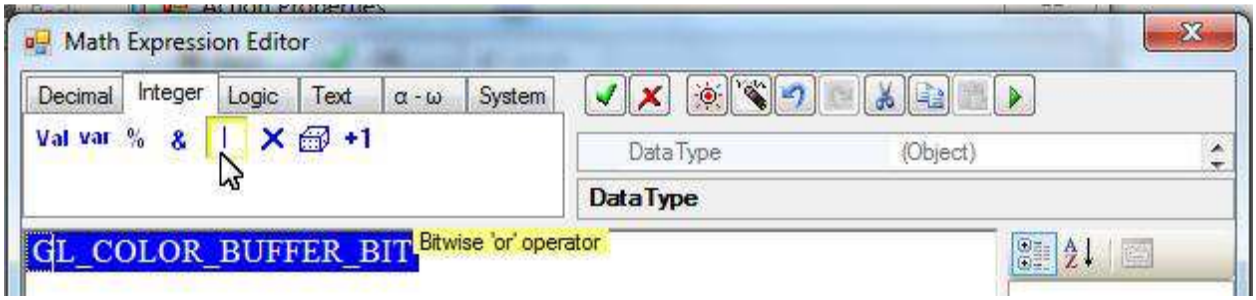
The mask definitions we are looking for are under the static fields:



Select GL_COLOR_BUFFER_BIT:



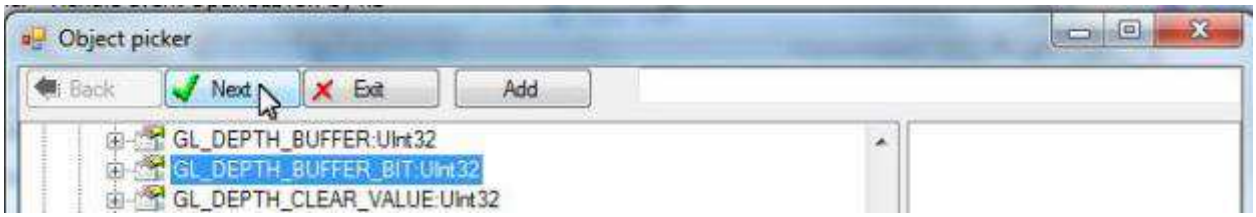
Click “bitwise or” operation:



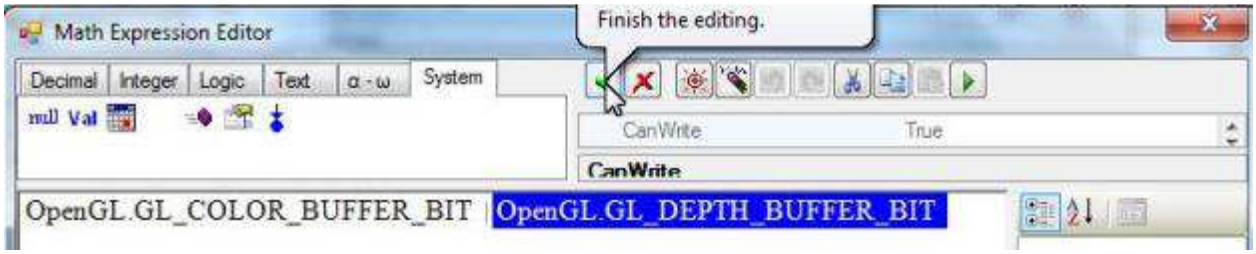
Click property icon to add another mask:

Click the value we selected previously to locate it so that we may quickly select the value near it:

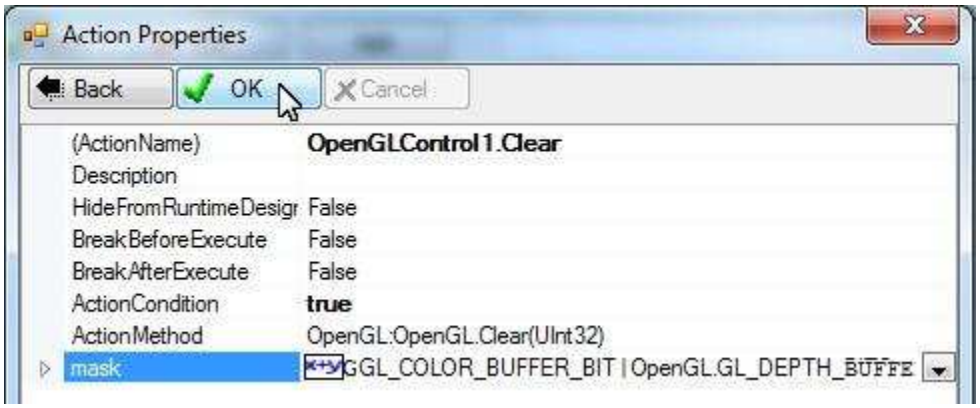
We want to add GL_DEPTH_BUFFER_BIT, it should be near the previously selected value:



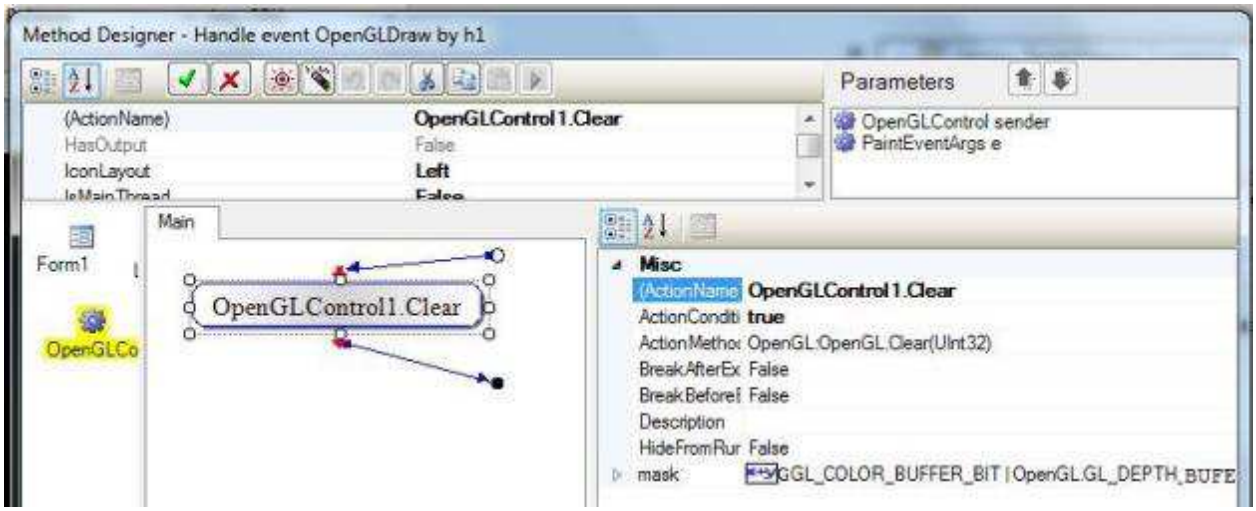
These are the masks for the action parameter:



Click OK.



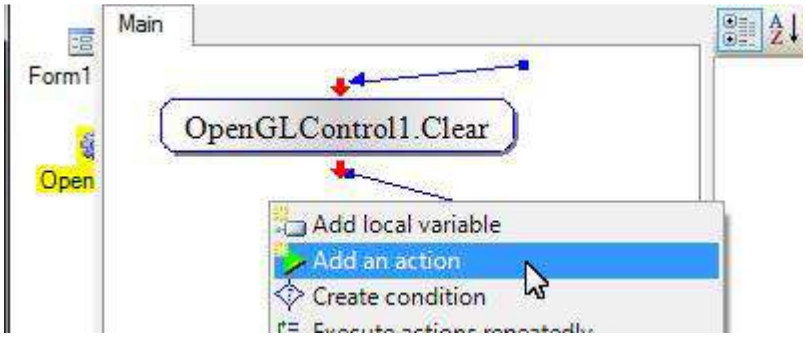
The action appears:



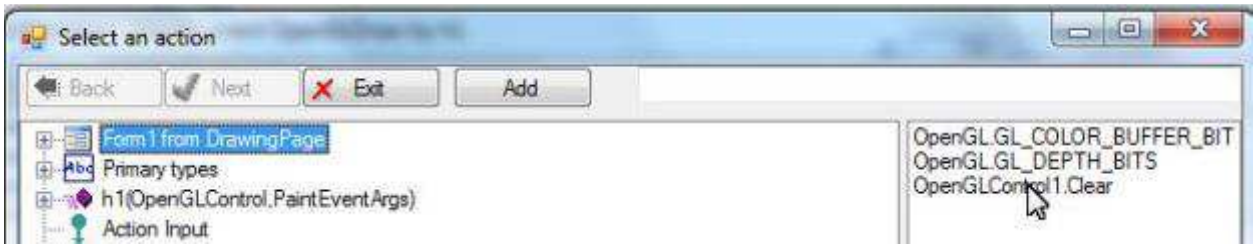
Draw pyramid

Prepare for drawing pyramid – reset coordinate transformations

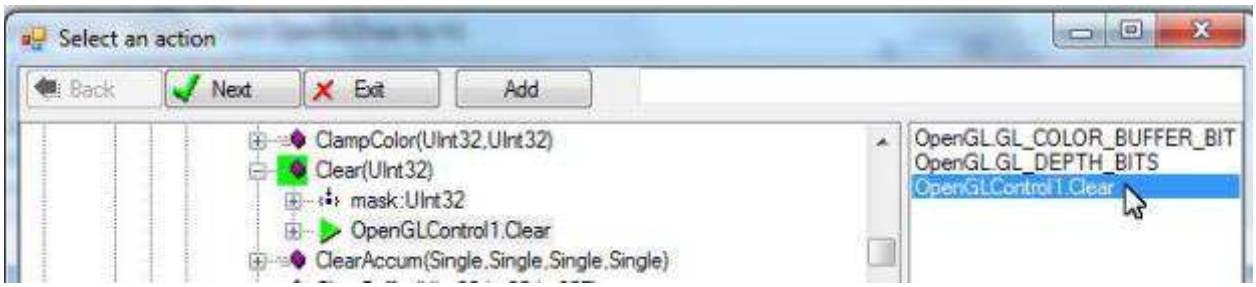
Add a LoadIdentity action to reset coordinate transformations. In OpenGL, all coordinate transformation actions are relative to current state. For example, if previously rotated 3 degrees then another rotating of 6 degrees will result in a total rotating of 9 degrees. If a LoadIdentity action is executed before the rotating of 6 degrees then the total rotating is 6 degrees not 9 degrees.



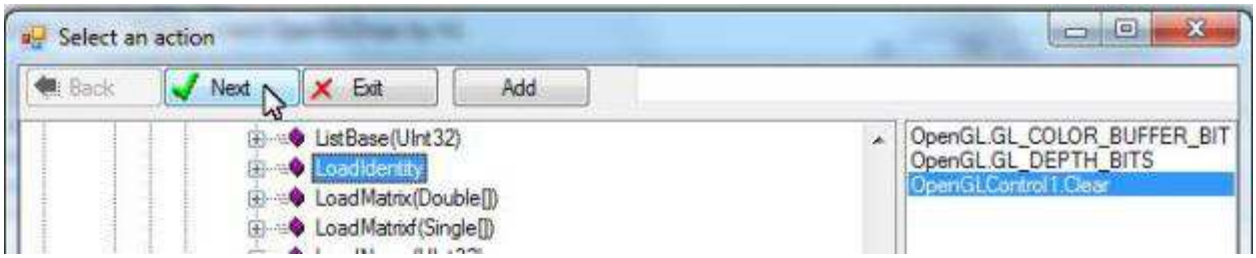
Click the previously used Clear method to help us to quickly locate OpenGL method, LoadIdentity:



It jumps to a previously created Clear action:



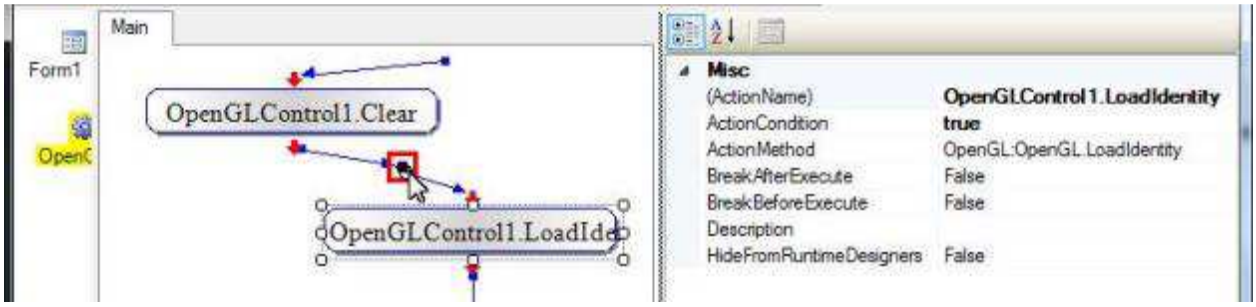
Scroll down to find LoadIdentity method:



Note that if the above quick location approach is not available, for example, Limnor Studio is restarted and the project is reloaded, then you may find the OpenGL methods under the OpenGL property of the OpenGLControl, like we did for locating Clear method.

Click OK to finish creating the action:

The action appears. Link it to the previous action:



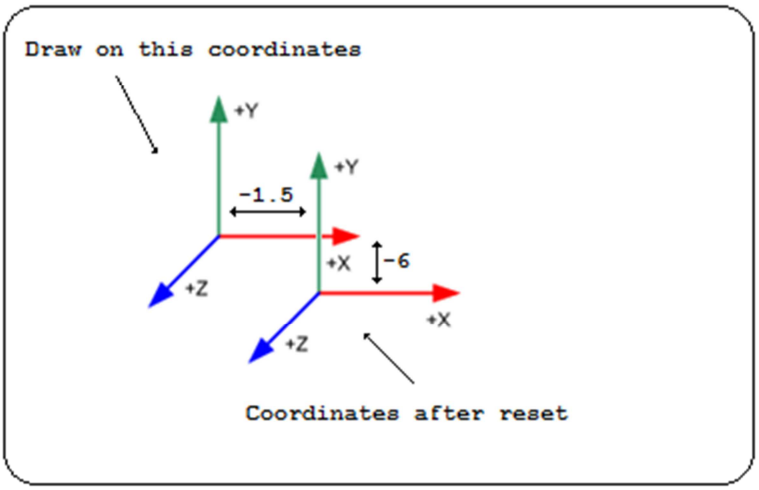
Prepare for drawing pyramid – set location

We want to draw the pyramid on the left side of the window and draw the cube on the right part of the window. So, we first use a Translation action to move the drawing location to the left side of the window.

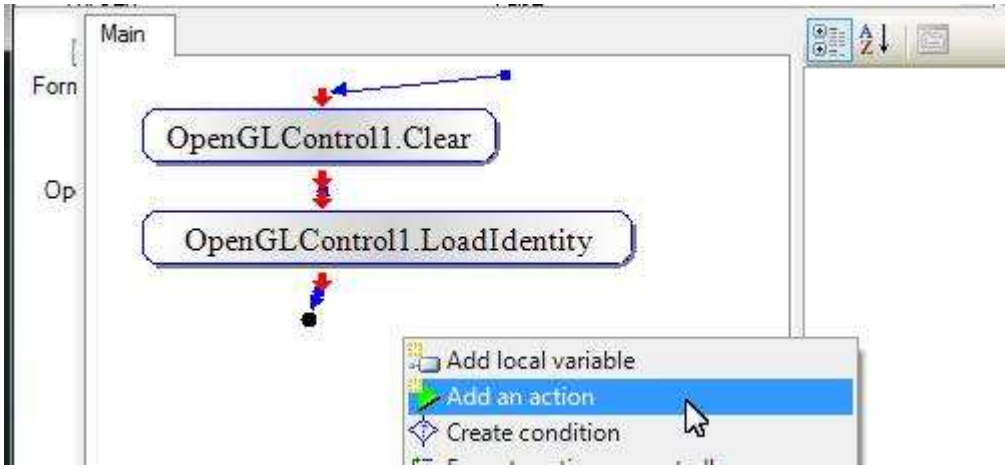
We do it by moving the x-axis by -1.5.

We also move z-axis by -6 to make the pyramid "inside" into the window to create a depth.

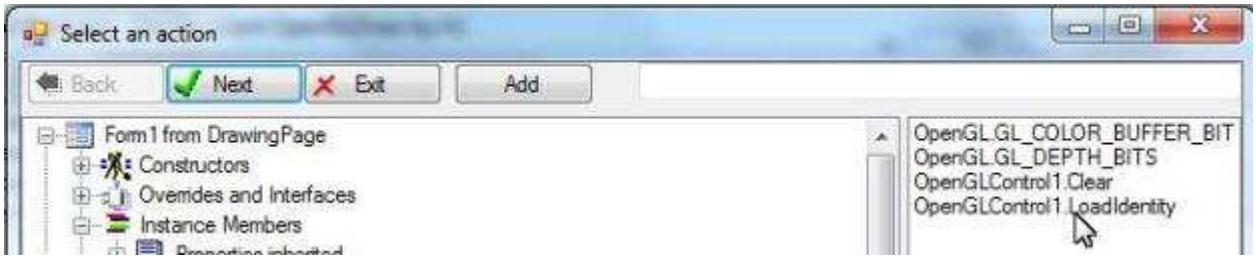
The effects can be illustrated by the following figure:

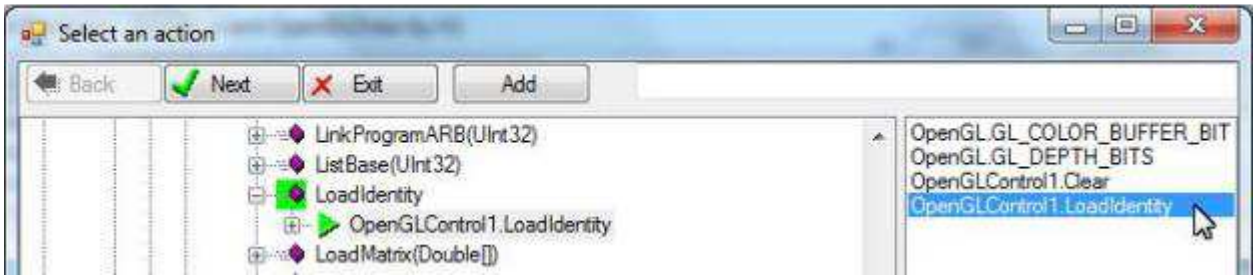


Let's create a Translate action to do it.

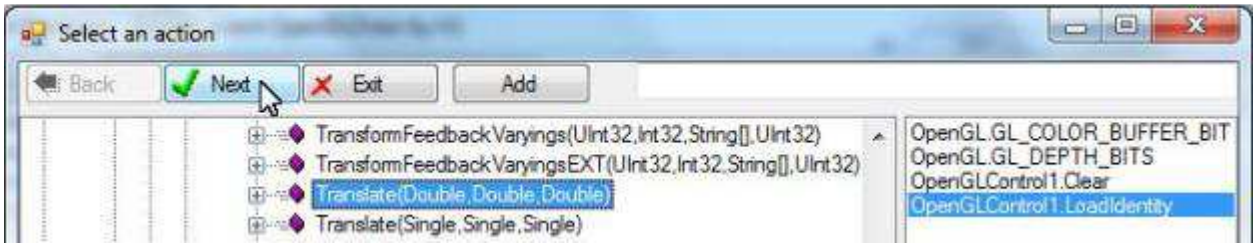


Click a previously used OpenGL method to help us locating the Translate method we want to use:

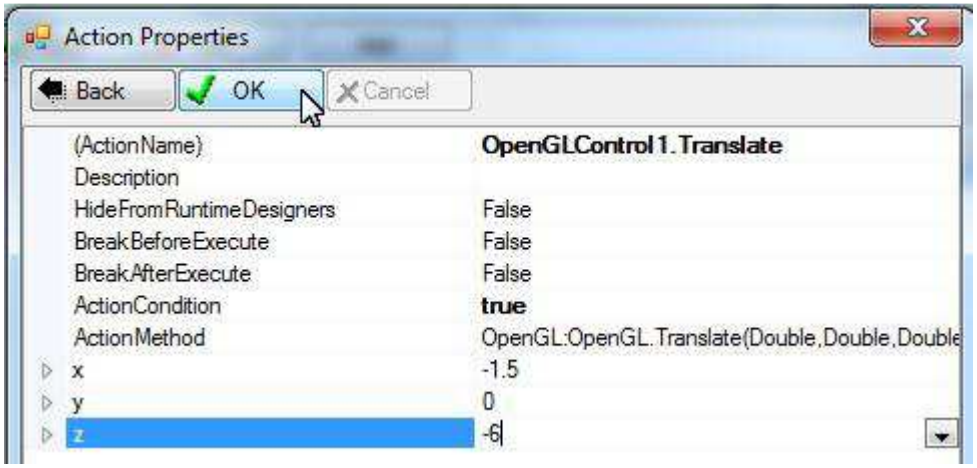




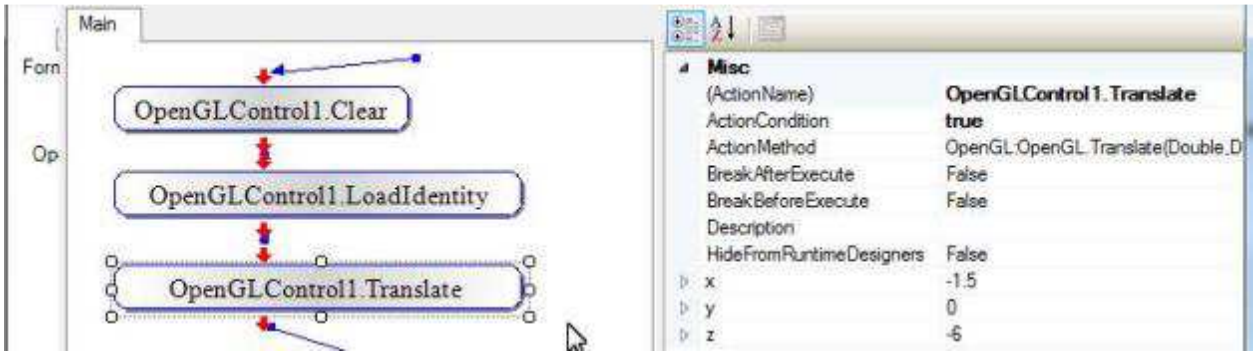
Select Translate method and click Next:



Set x-axis moving amount to -1.5 and z-axis moving amount to -6:



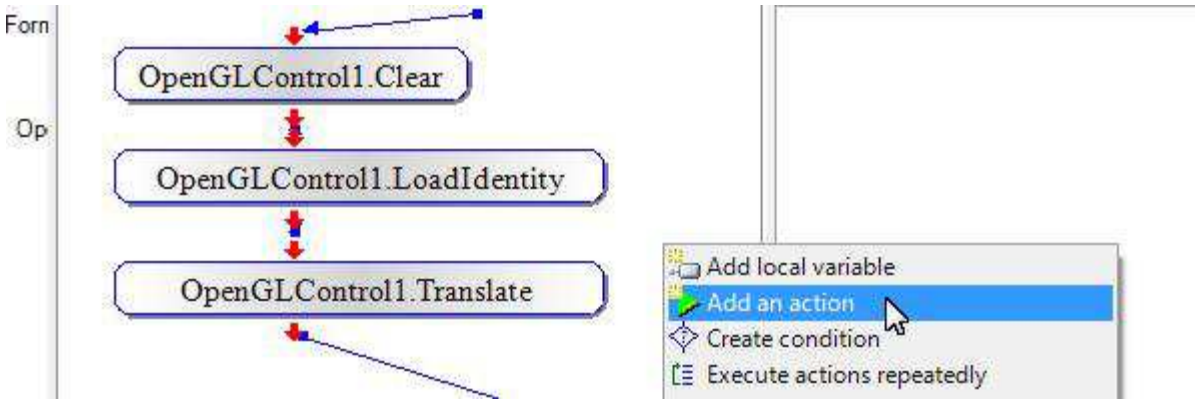
Link it to the last action:



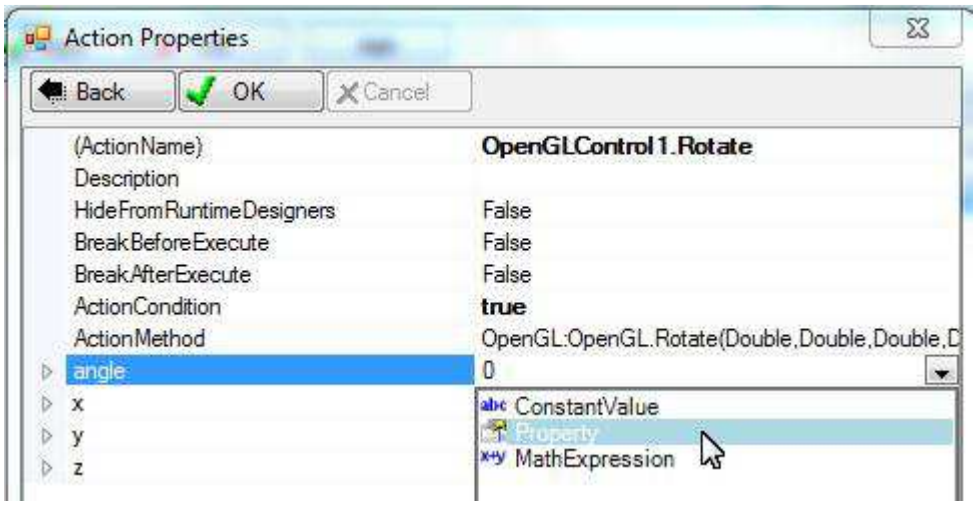
Prepare for drawing pyramid – rotate

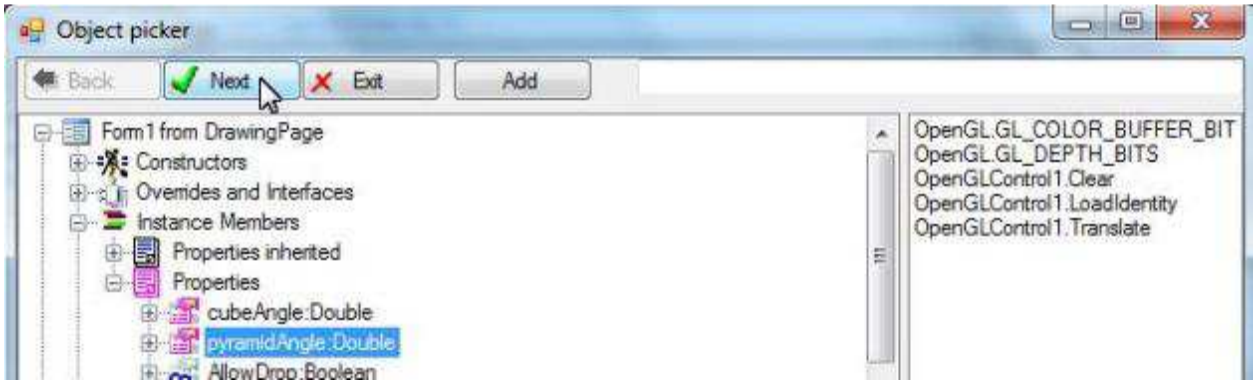
We are drawing one frame in handling one OpenGLDraw event. We need to rotate it using the property pyramidAngle. After drawing the frame, we will increase the rotate angle so that each frame will use a different angle to create a rotating effect.

Let's create a Rotate action to rotate the coordinate.

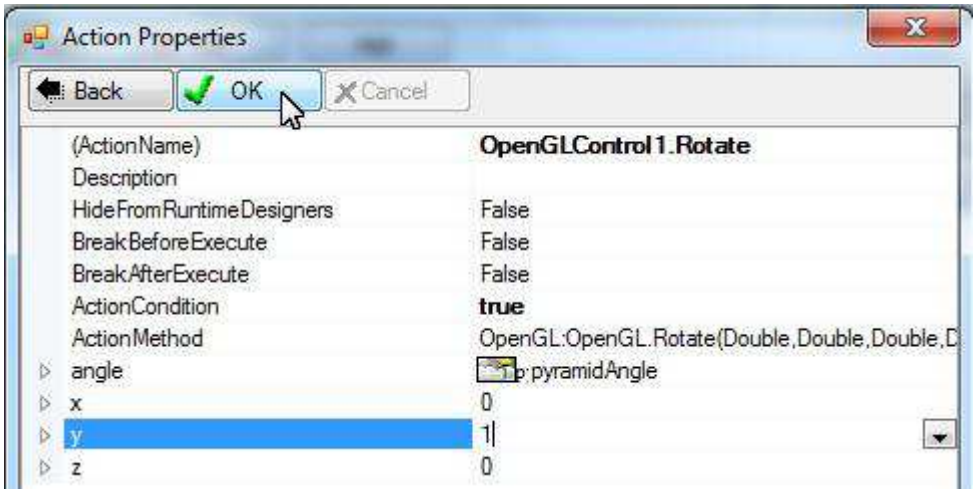


Select Property for "angle" parameter of the action to use pyramidAngle property:

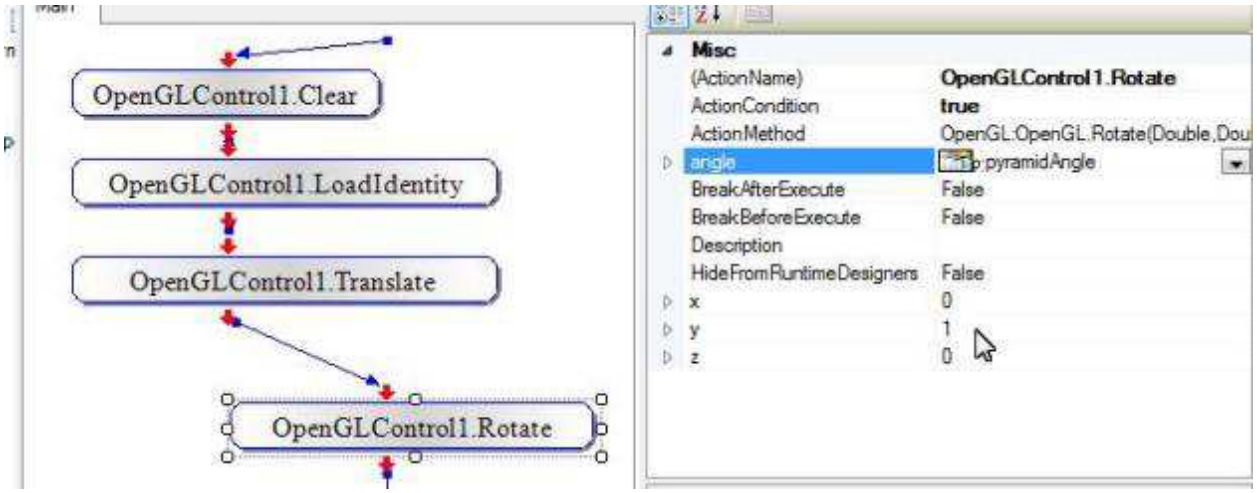




Set 1 to parameter "y" to indicate that the rotation is on y-axis:

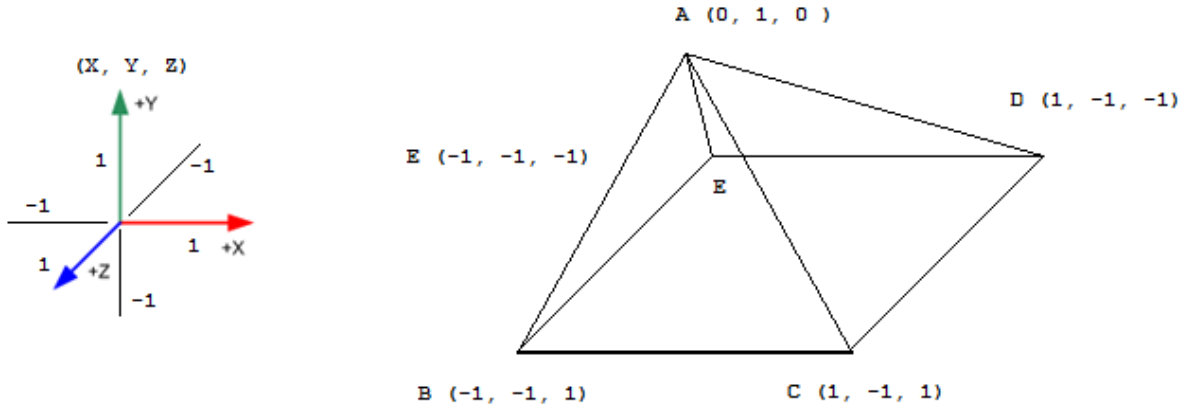


Link the action:



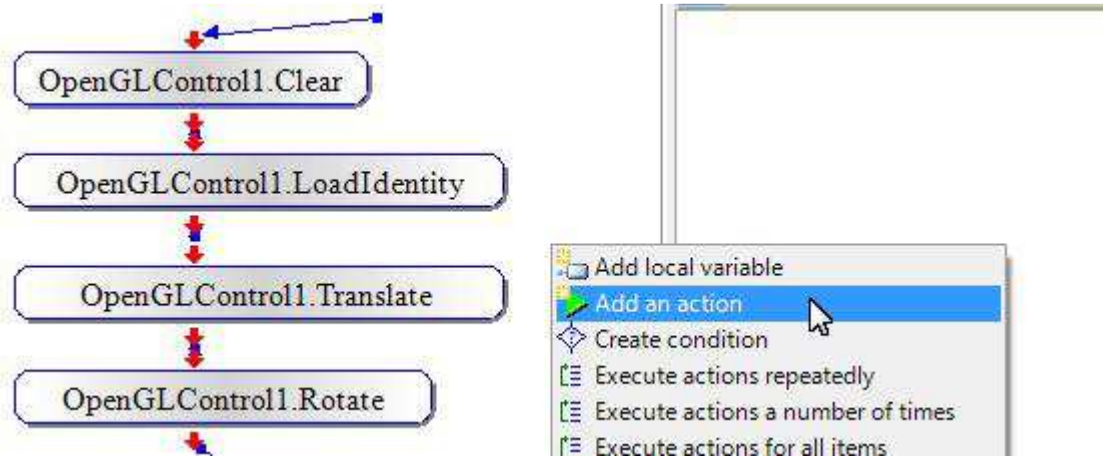
Draw pyramid sides

A pyramid consists of 4 triangle sides: ABC, ACD, ADE, and AEB. We need to draw these 4 triangles.

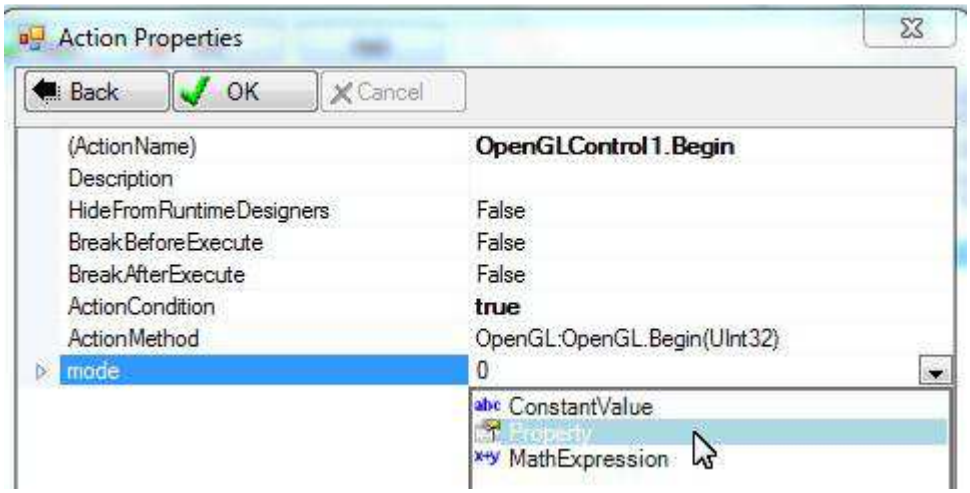


Begin drawing triangles

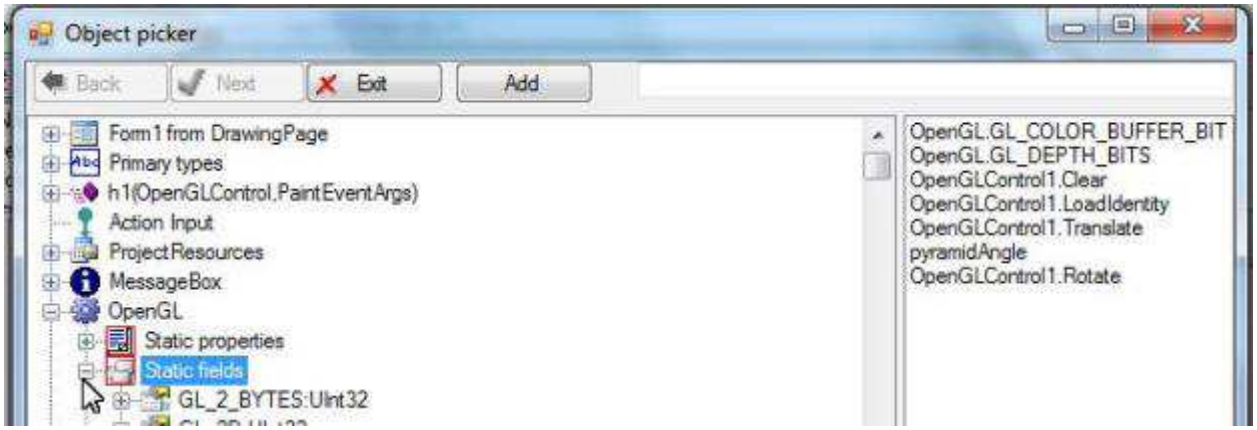
The first step in drawing a pyramid is to execute a **Begin** action and indicate that we want to draw triangles.

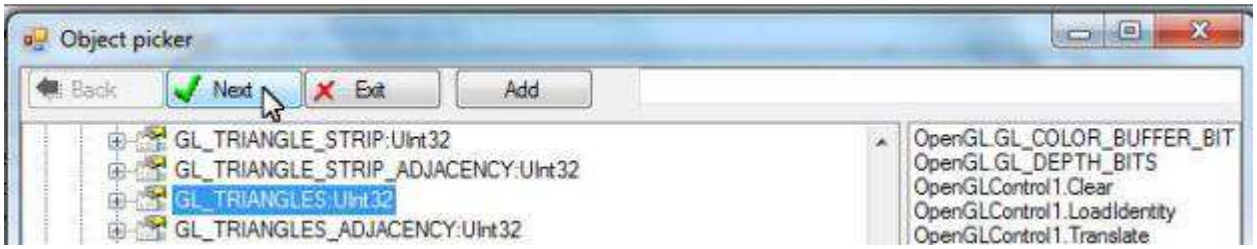


Select Property to locate mode for triangles:

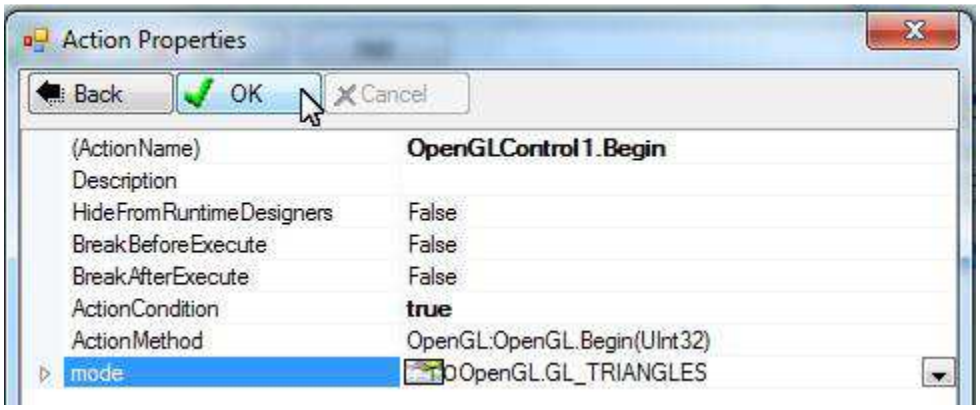


The mode can be found in under the “static fields” listing:

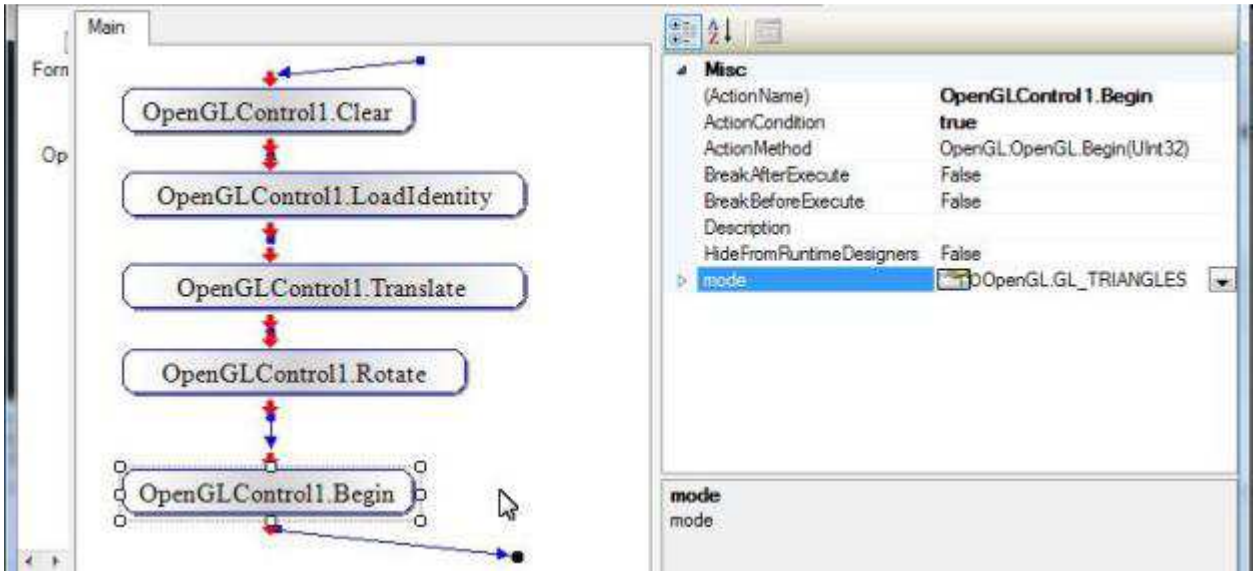




Click OK:

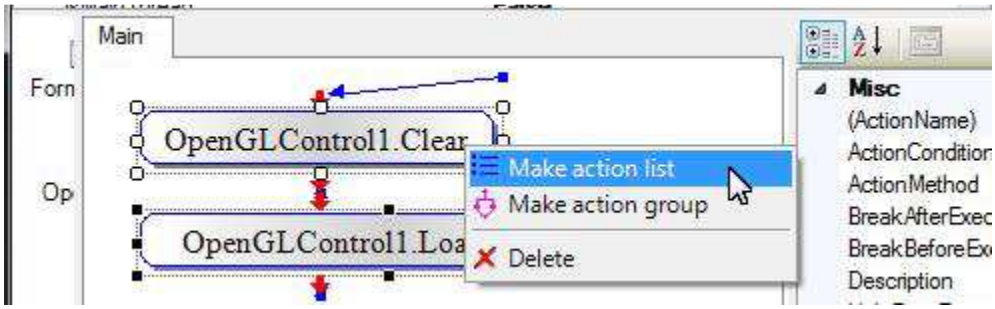


Link it to the last action:

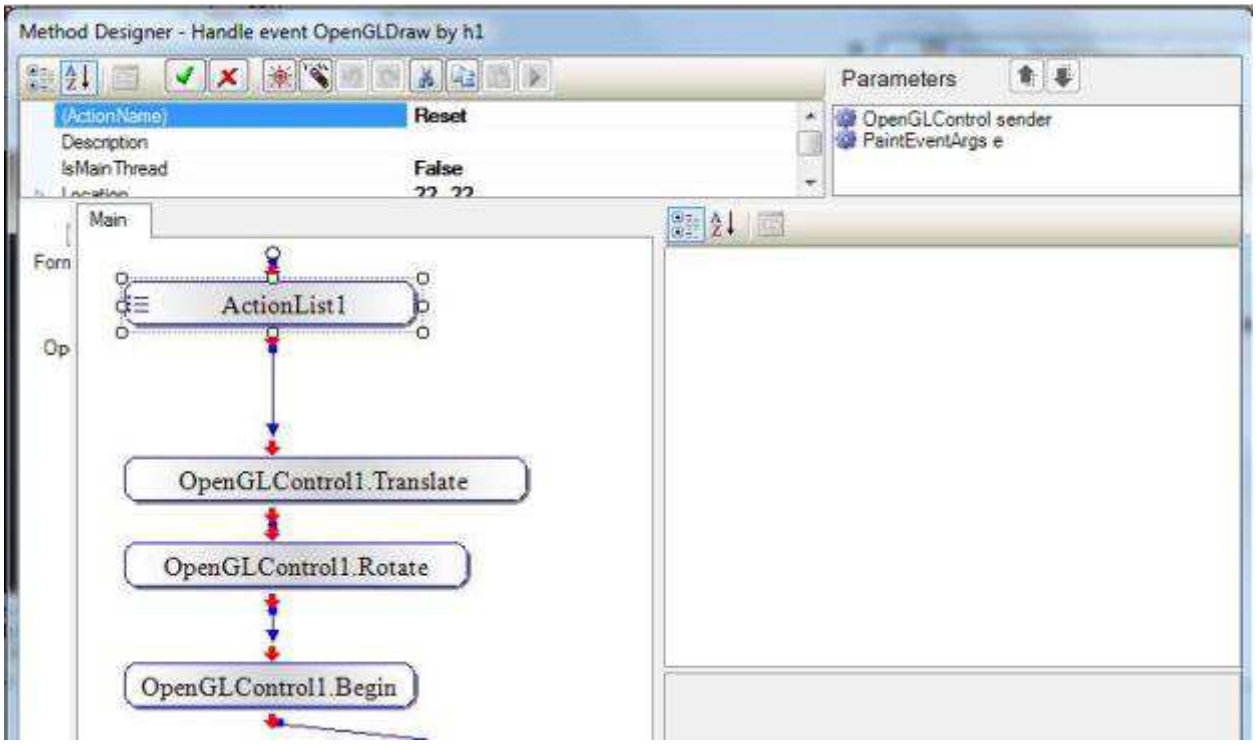


We can see that the method editor screen is getting messy with more and more actions. We may group the actions into action lists and action groups to make it cleaner. In our sample here, we use action lists.

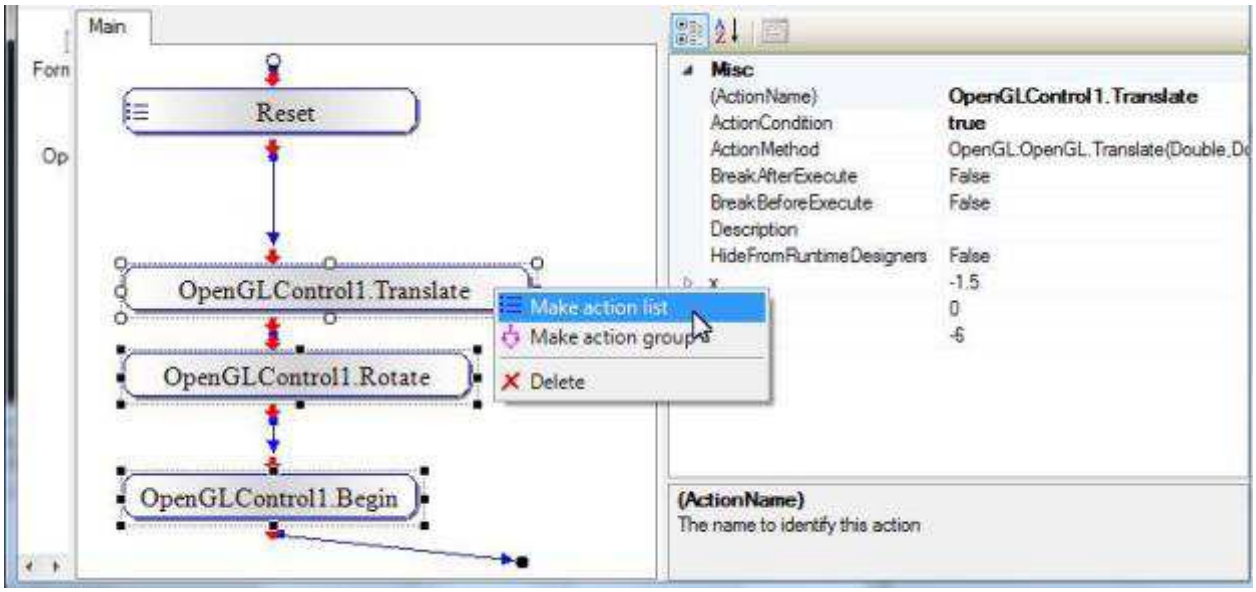
Group the first 2 actions into an action list:



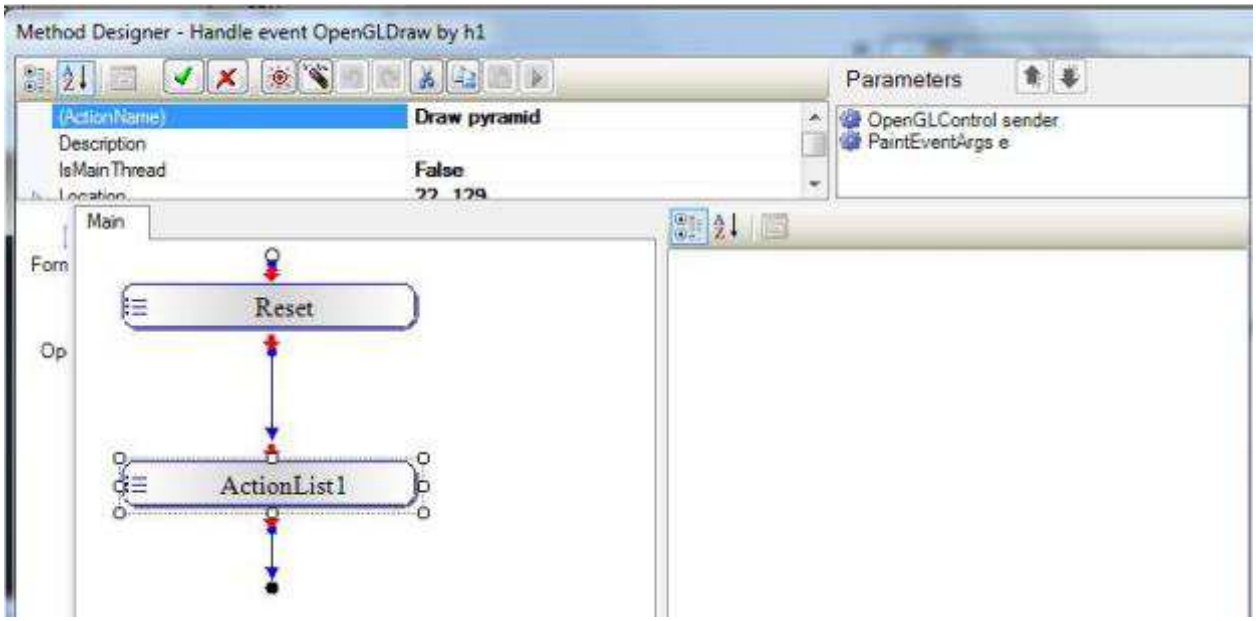
Name it “Reset”:



Group the next 3 actions into another action list:



Name it “Draw pyramid”:



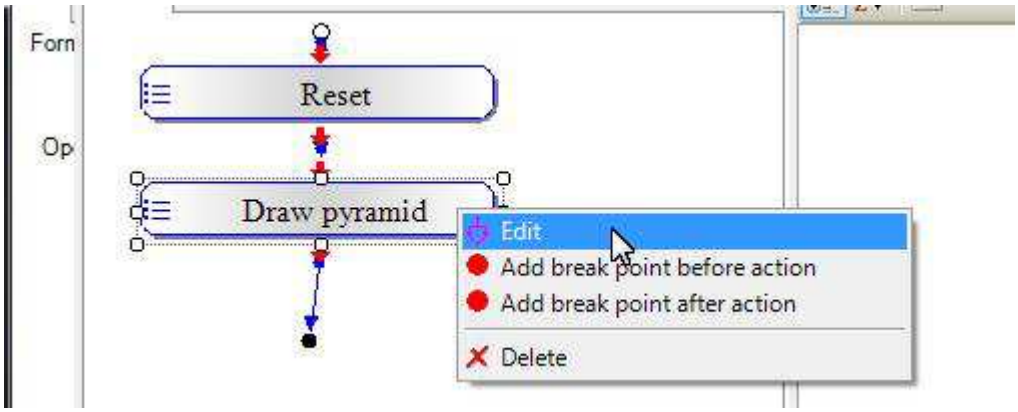
Draw triangle ABC

Draw triangle ABC is to use Vertex actions for point A, B, and C, respectively.

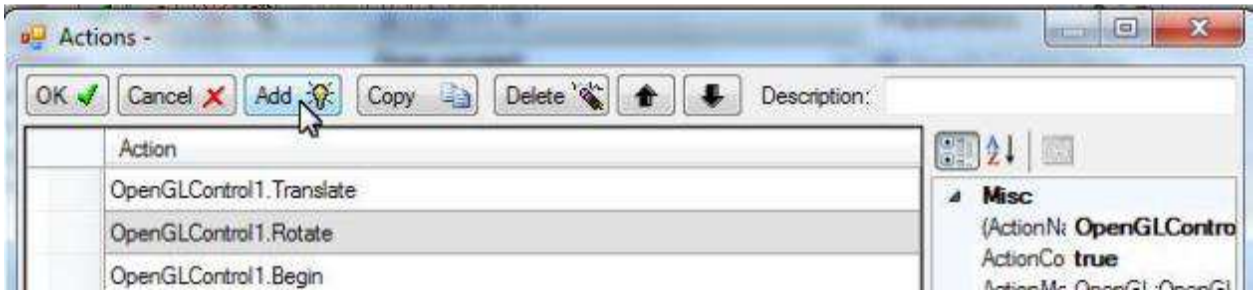
Draw vertex A -- Set drawing color to red

We use a Color action to prepare drawing color so that vertex A is drawn with red color.

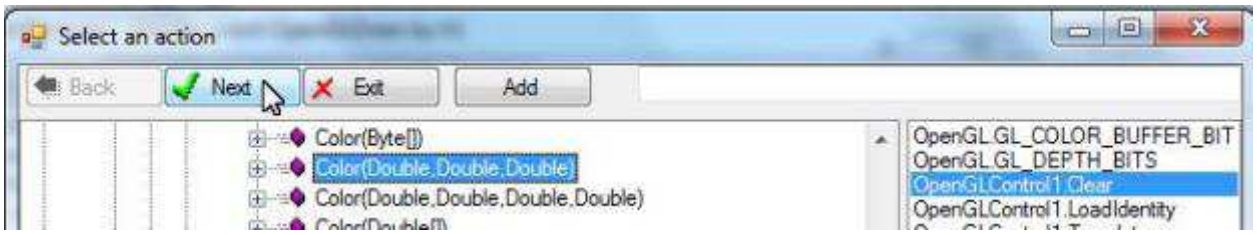
We will add all actions for drawing the pyramid in the action list “Draw pyramid”. Right-click the action list and choose “Edit”:



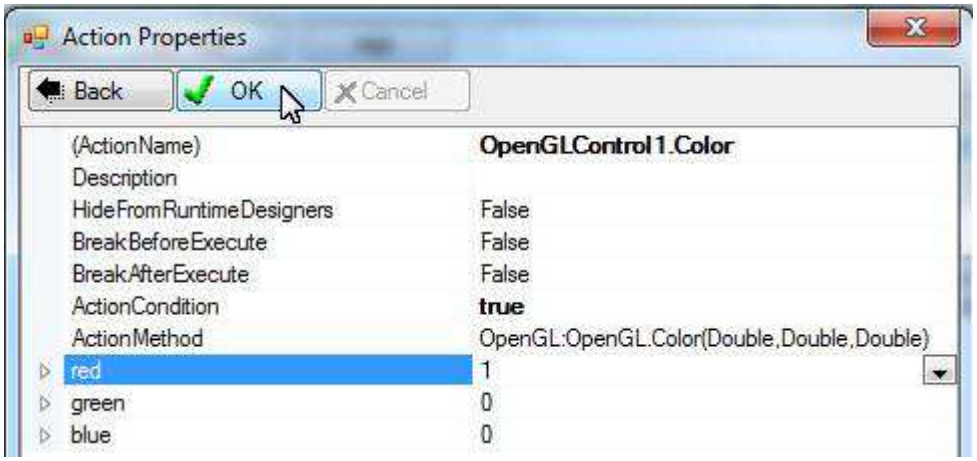
Click Add button to add a new action to the list:



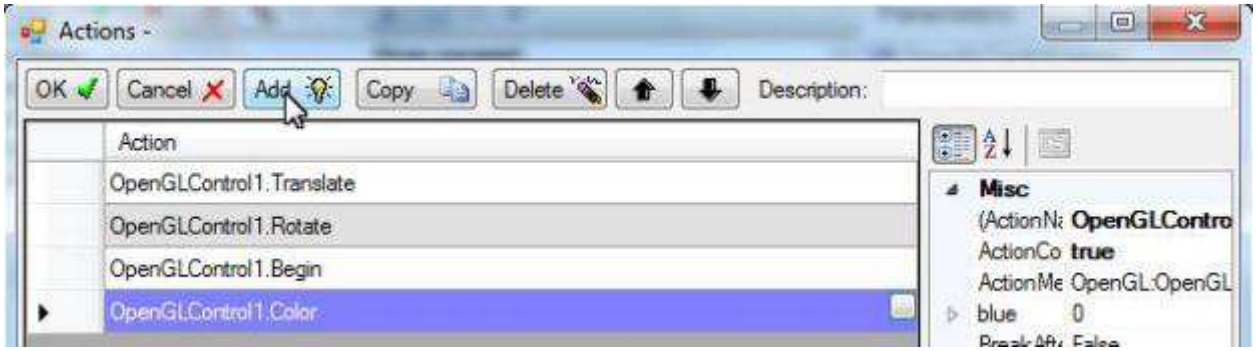
Select Color method:



To use red color, set “red” parameter to 1:



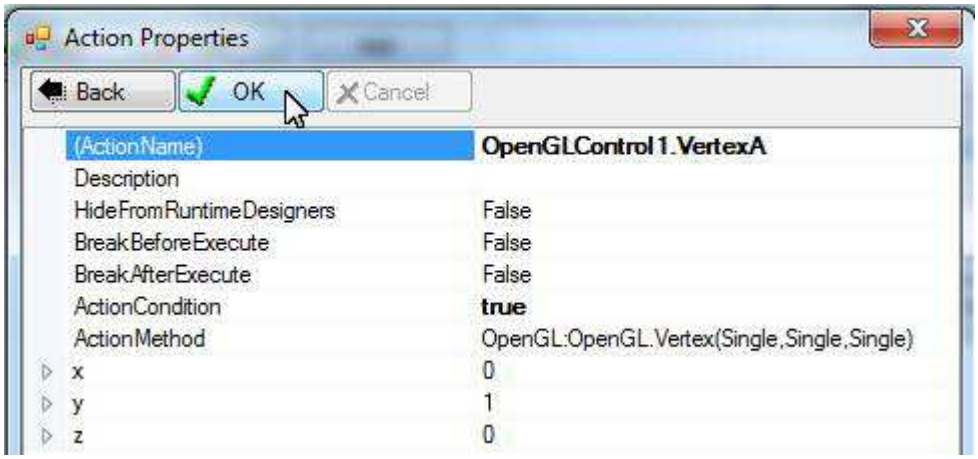
The action appears at the end of the list. Click Add button again to add a Vertex action for point A:



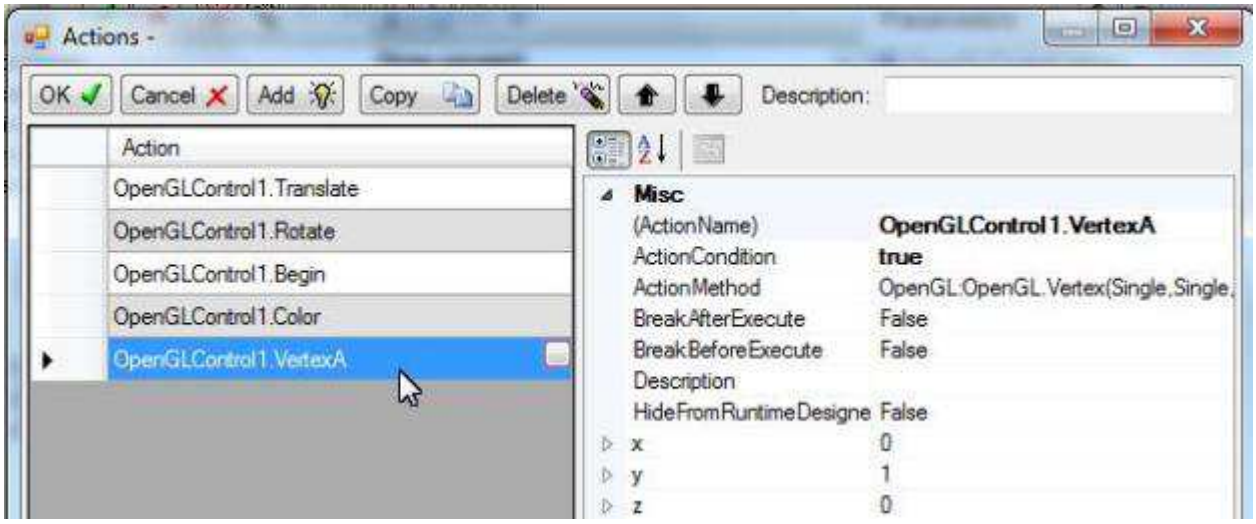
Draw vertex A – execute Vertex action

Select Vertex method:

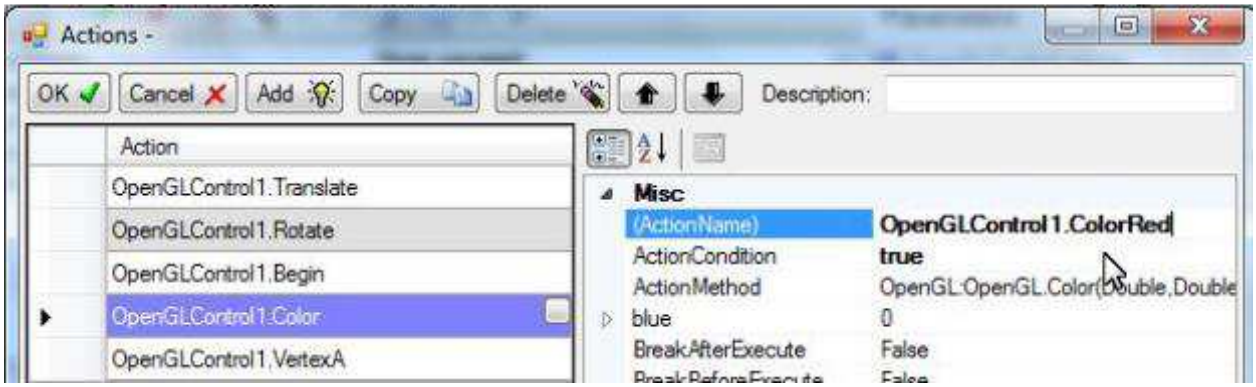
Set the coordinates for point A (0, 1, 0):



The action appears at the end of the list:



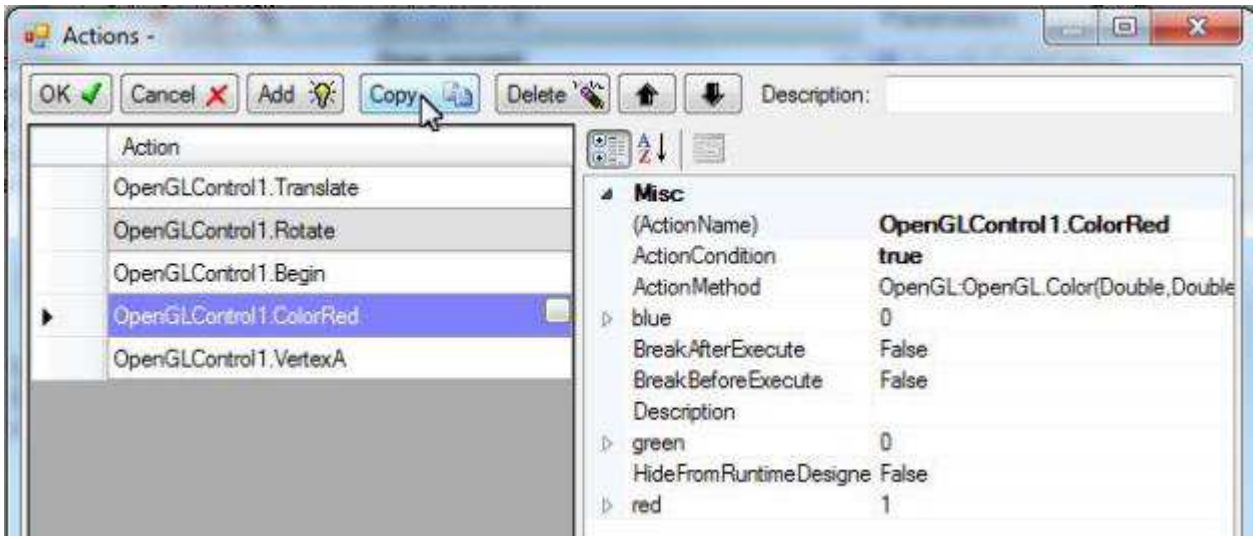
Let's change the name of action OpenGLControl1.Color to OpenGLControl1.ColorRed to make it clear what the action does:



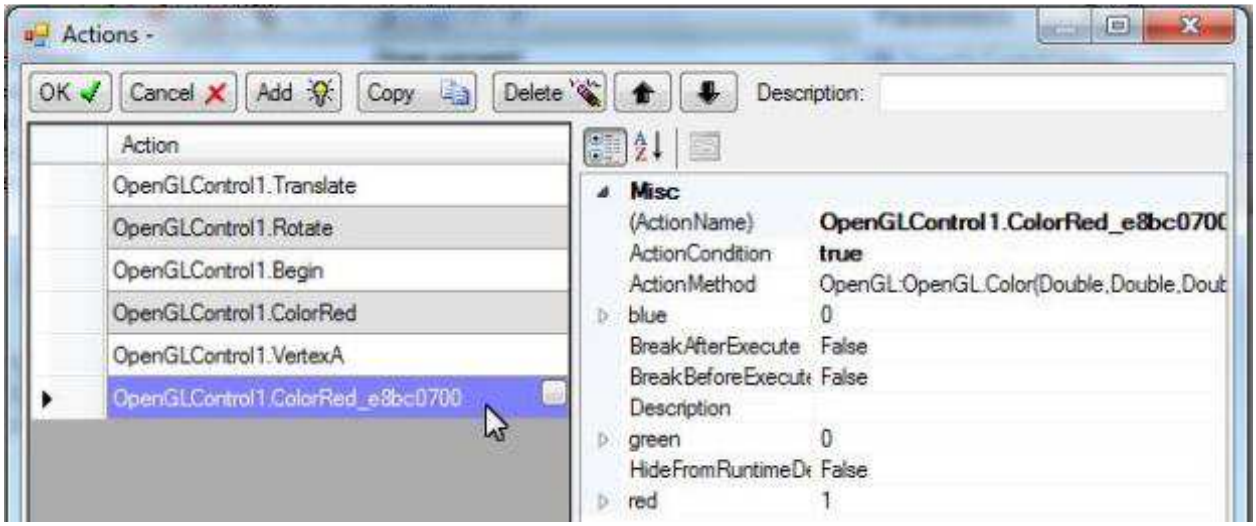
The two actions, OpenGLControl1.ColorRed and OpenGLControl1.vertexA, draw vertex A of the triangle ABC.

Draw vertex B of triangle ABC

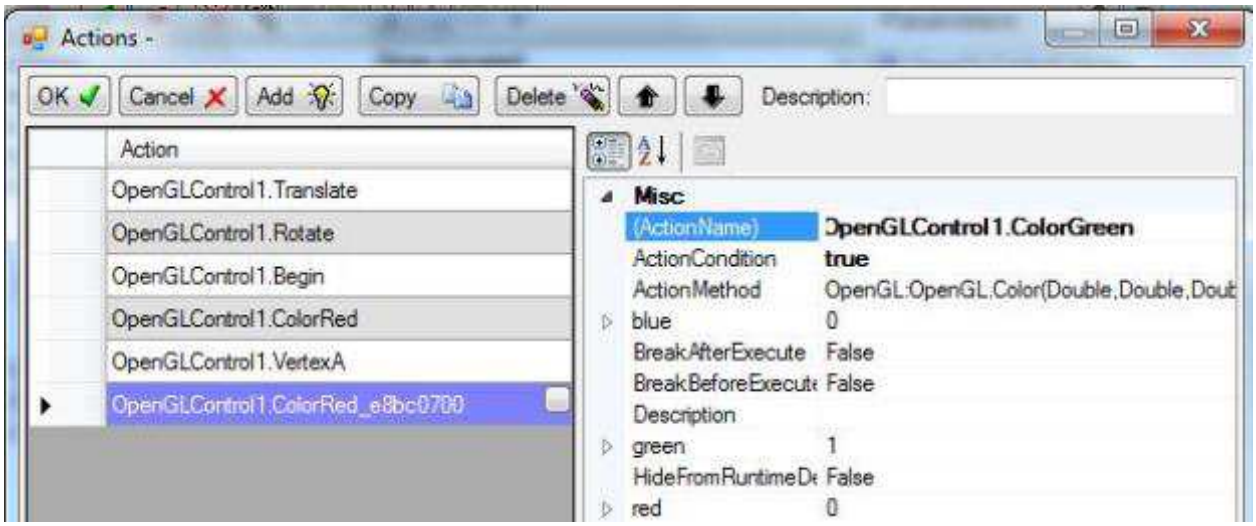
Let's draw this vertex with color green. Let's copy action OpenGLControl1.ColorRed:



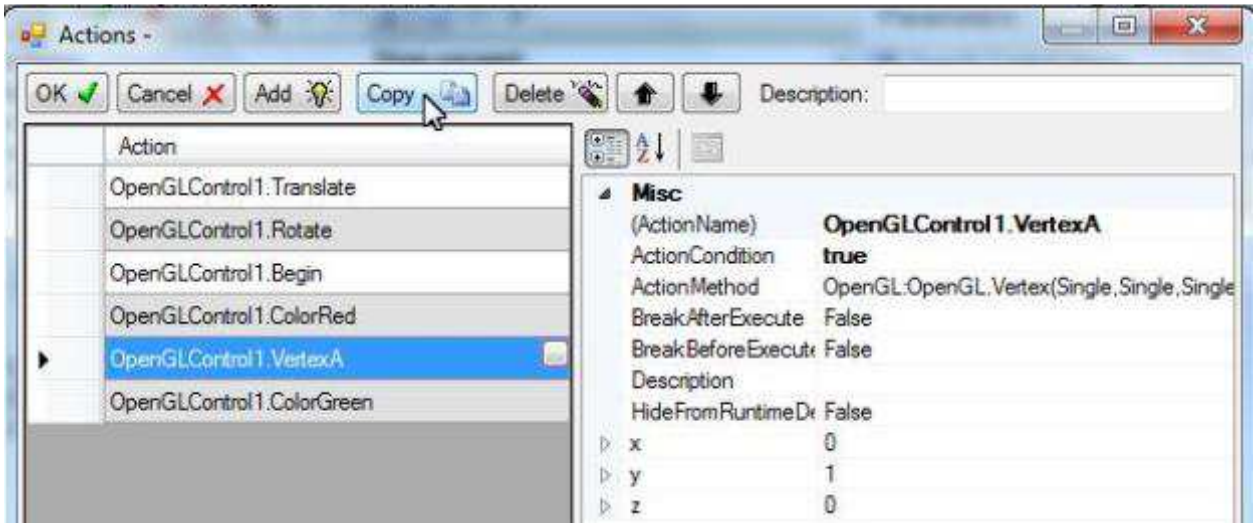
A new copy of Color action is created and appended to the end of the list:



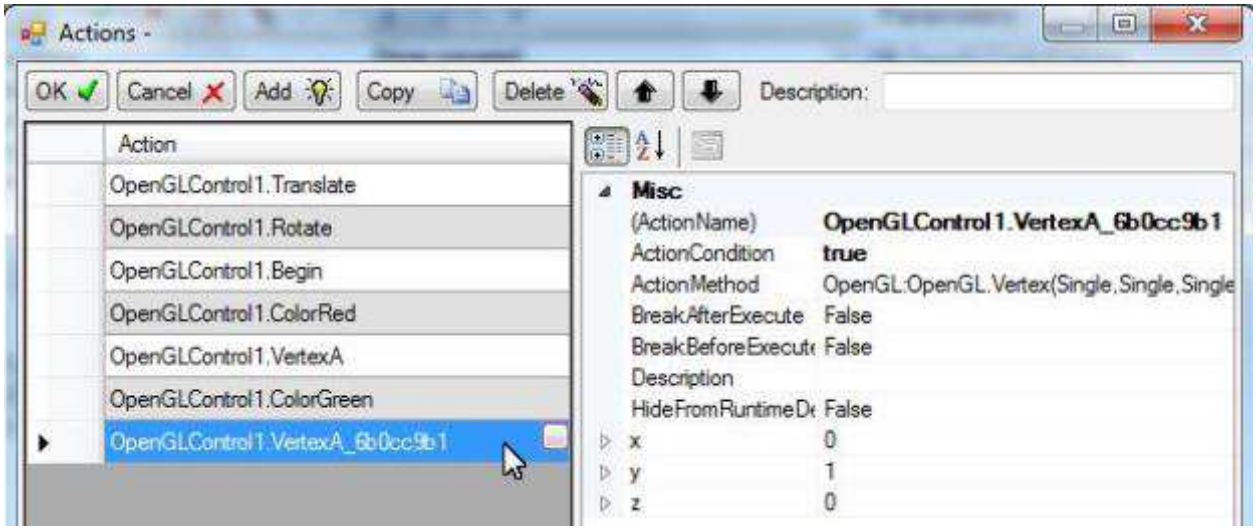
Change “green” to 1 and “red” to 0. Rename the action to OpenGLControl1.ColorGreen:



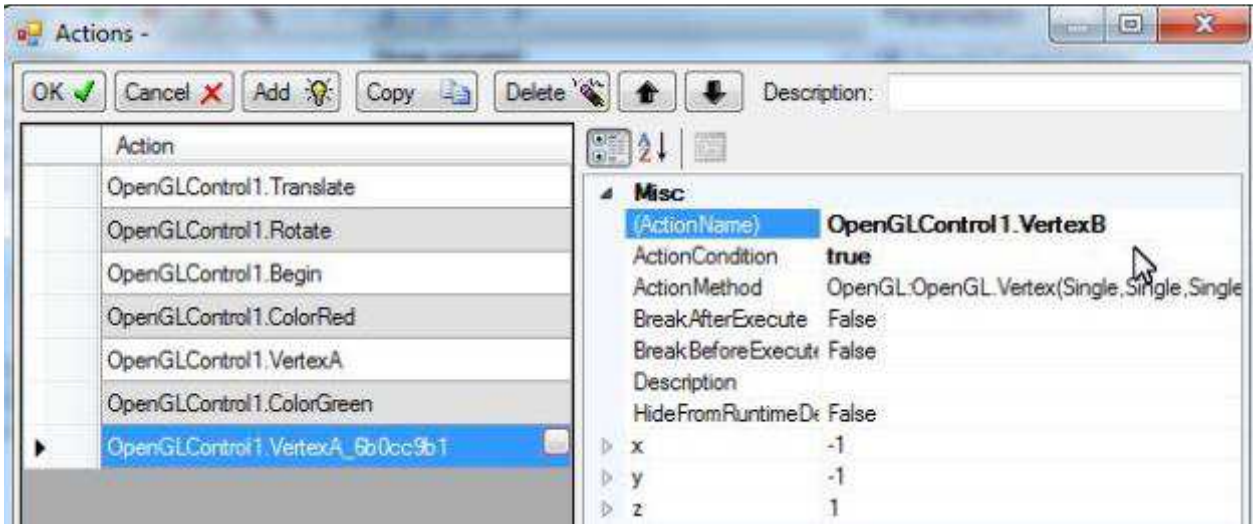
Copy action OpenGLControl1.VertexB:



A new copy of Vertex action is created and appended to the end of the list:

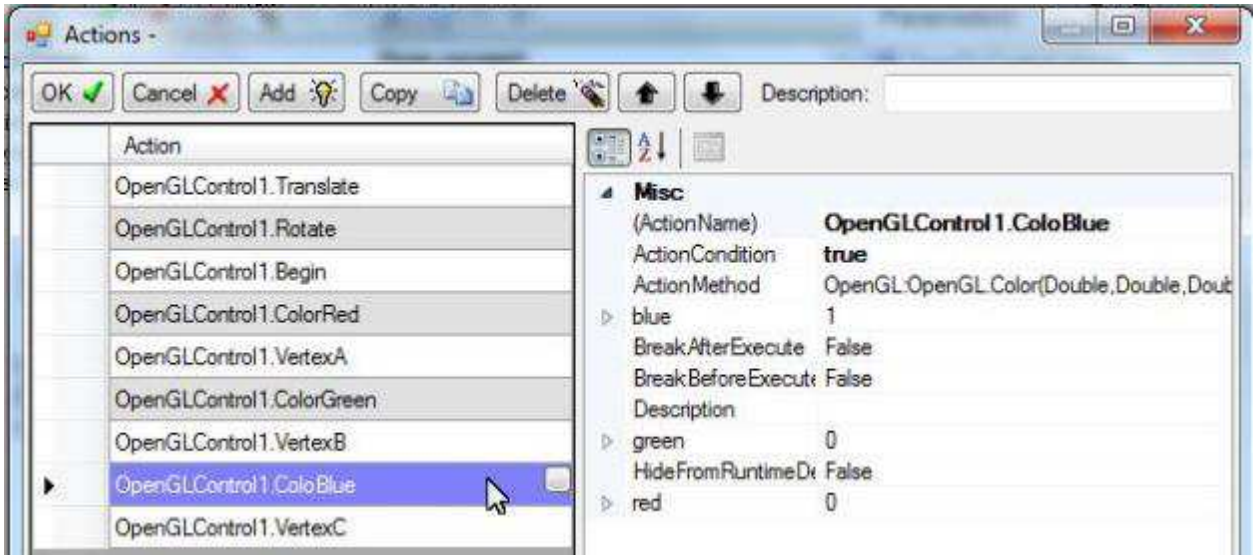


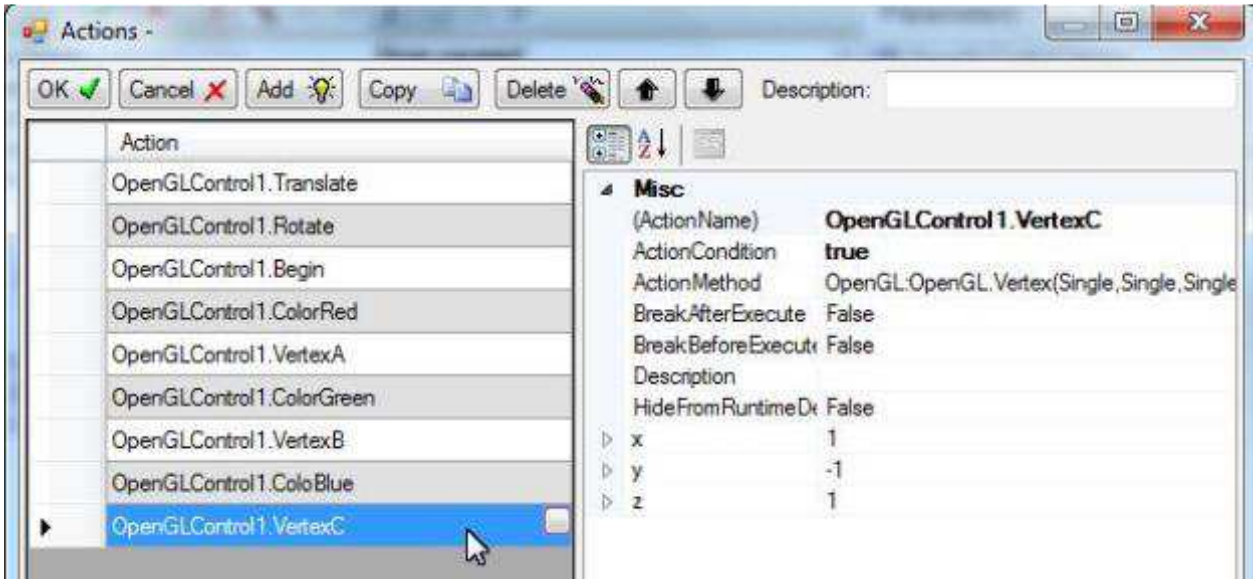
Apply the coordinates of point B (-1, -1, 1) to the action and rename it to OpenGLControl1.VertexB:



Draw vertex C of triangle ABC

We'll use color blue to draw vertex C of the triangle ABC. Using the copying and modifying as we did for vertex B, we create another two actions, OpenGLControl1.ColorBlue and OpenGLControl1.VertexC:





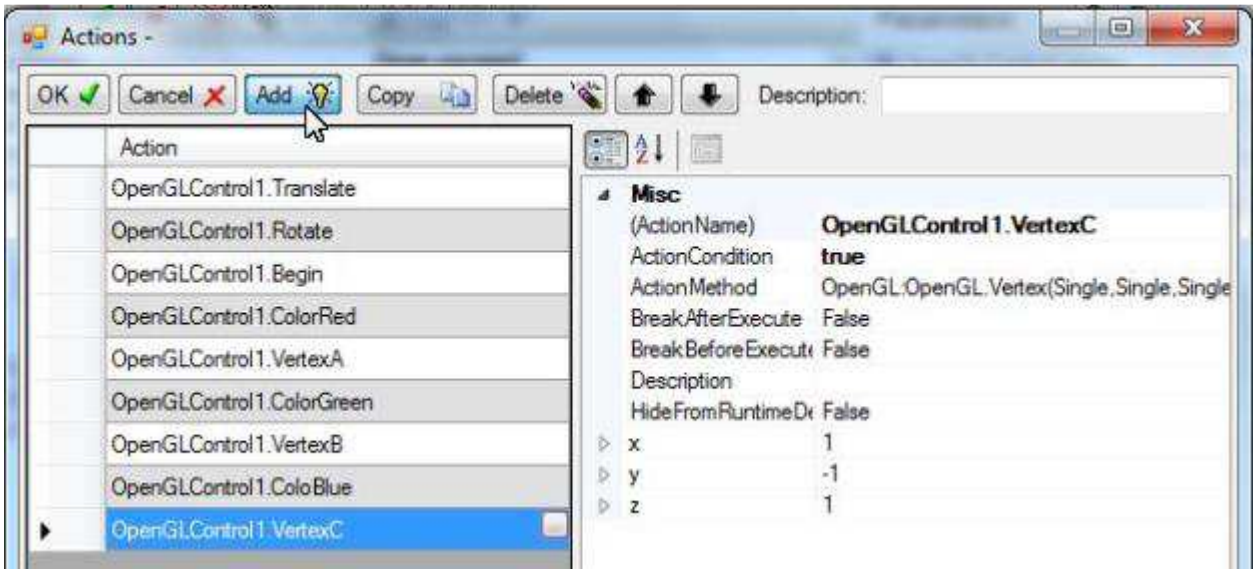
We thus finish drawing triangle ABC.

Draw triangle ACD

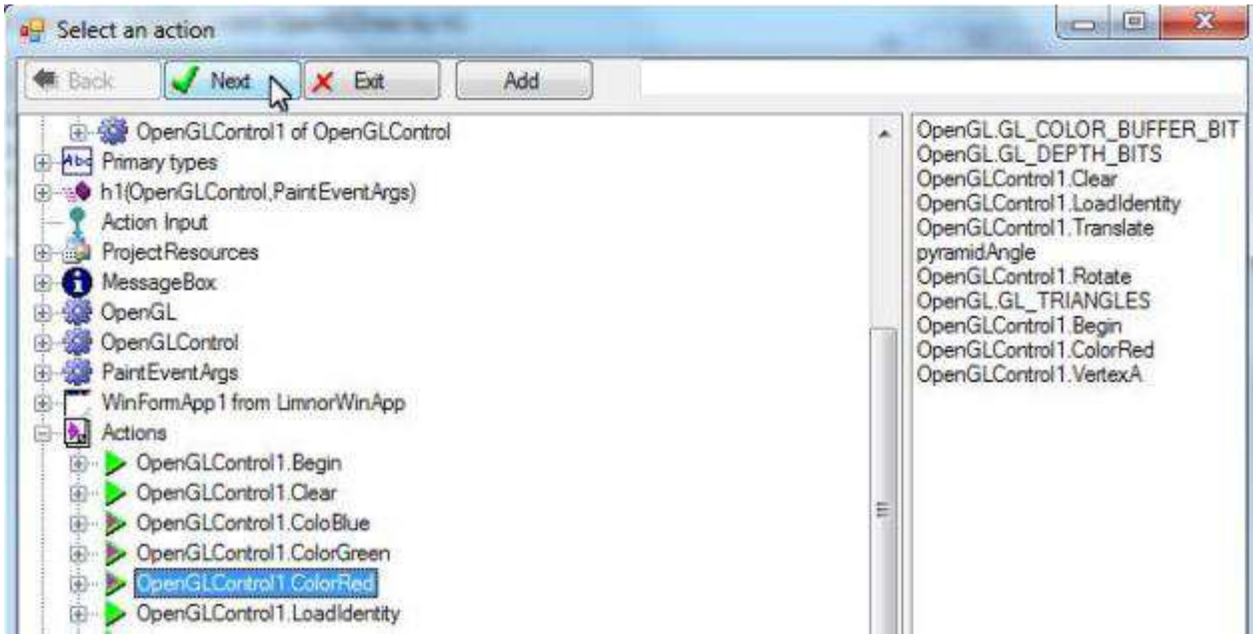
Draw triangle ABC is to use Vertex actions for point A, C, and D, respectively.

Drawing vertex A with red

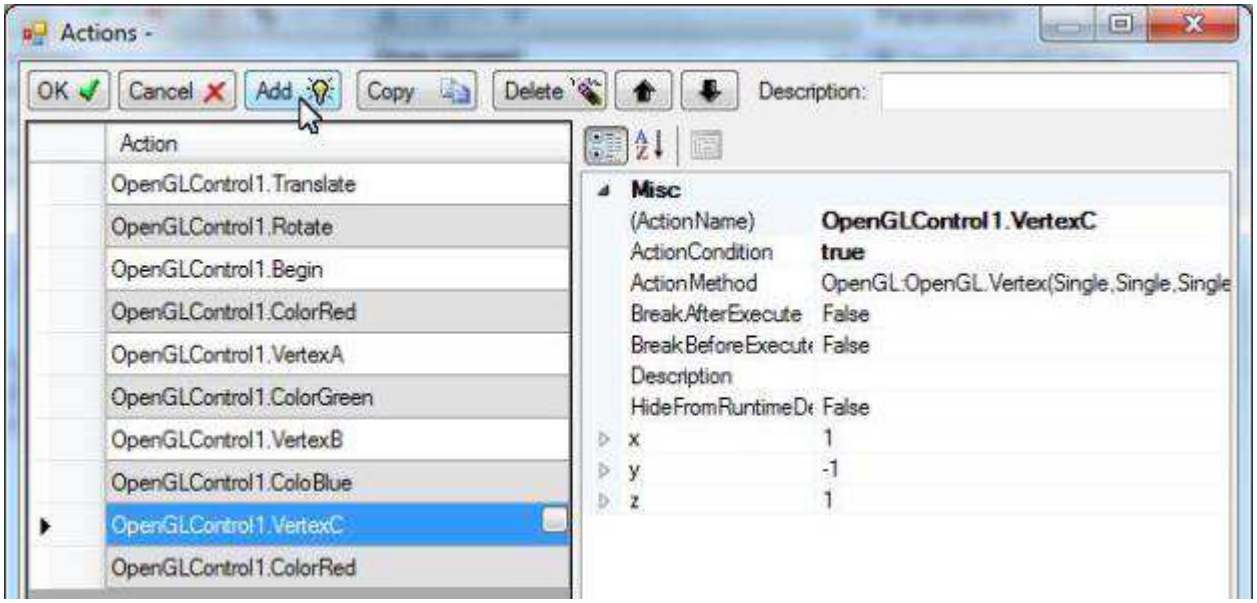
We already did it by actions OpenGLControl1.ColorRed and OpenGLControl1.VertexA. We just add these two actions to the list again:



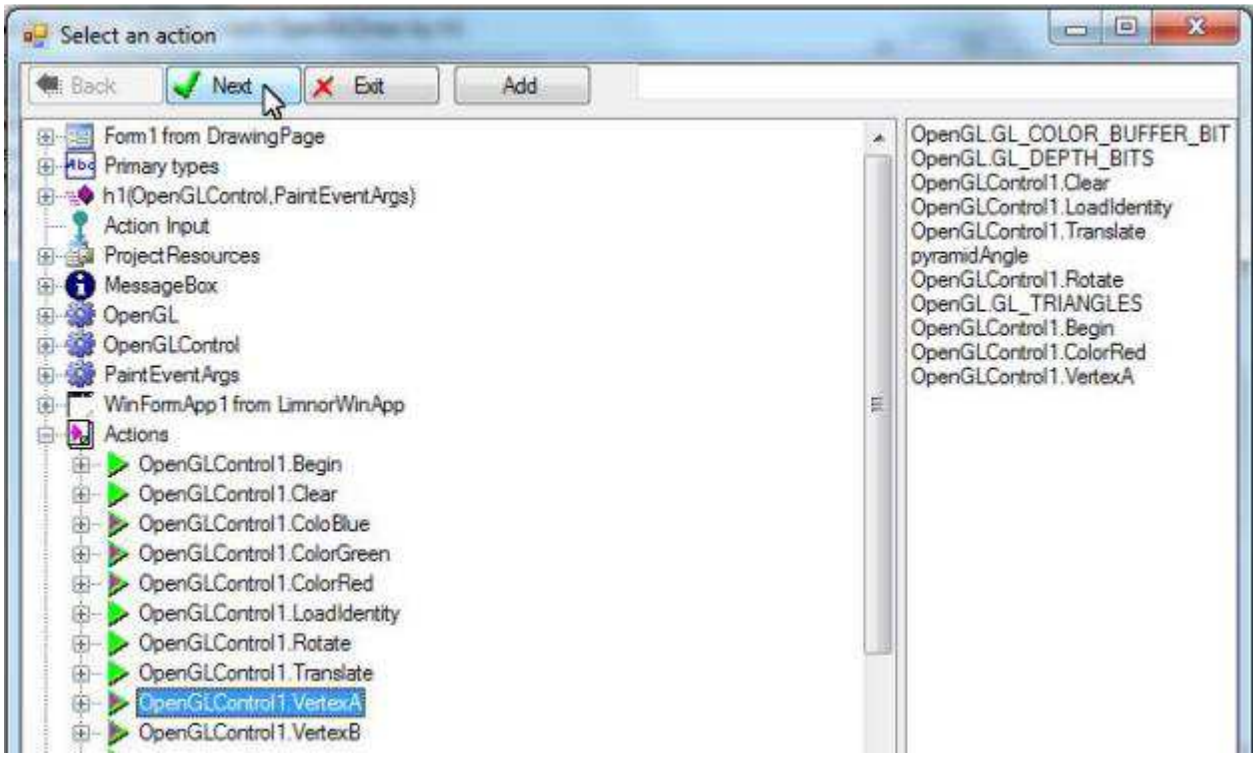
Select action OpenGLControl1.ColorRed:



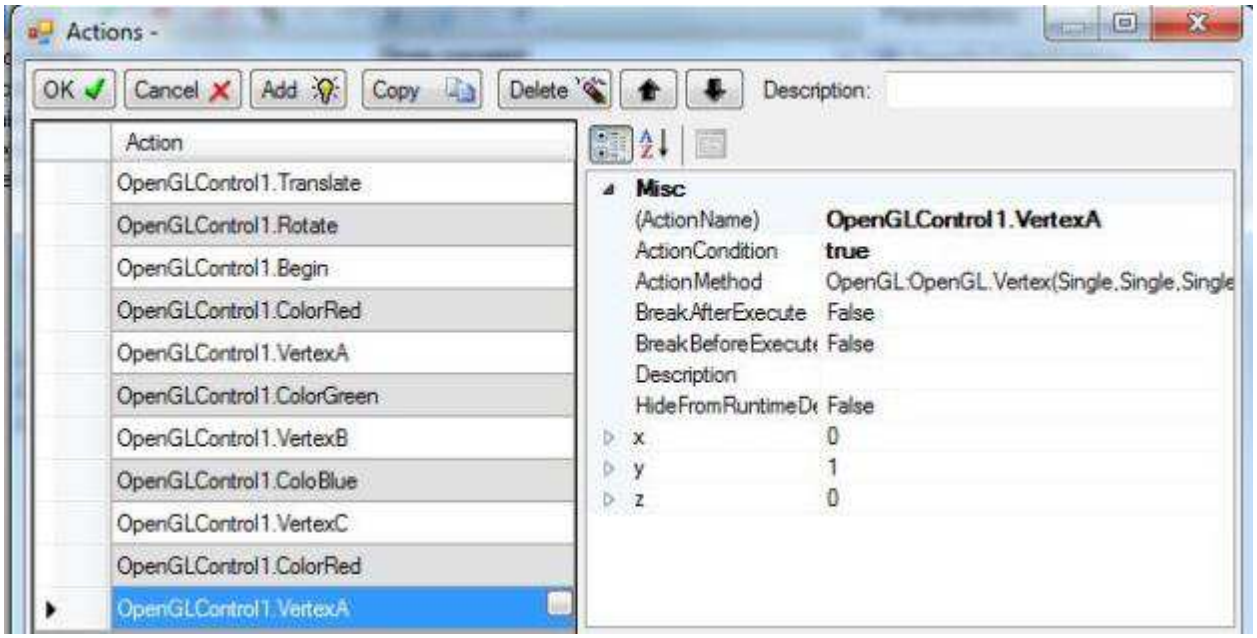
The action appears at the end of the list. Click Add again:



Select action OpenGLControl1.VertexA:

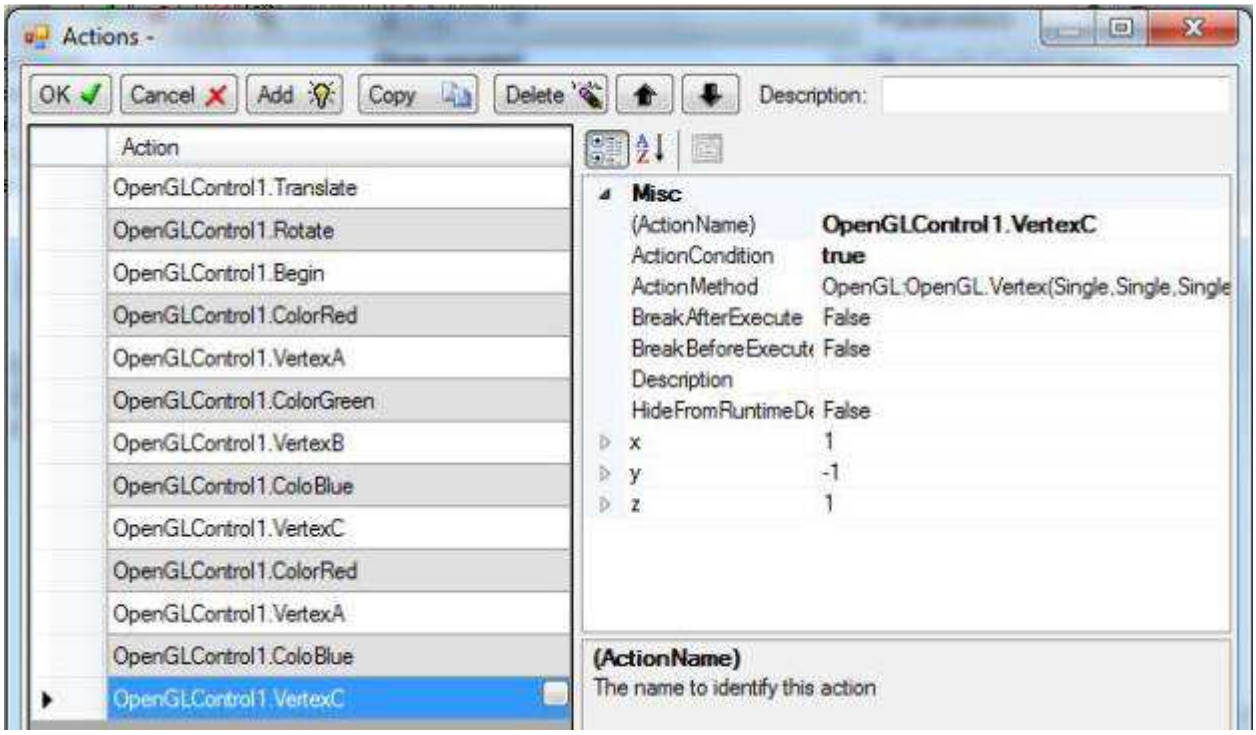


The action appears at the end of the list:



Draw vertex C with blue

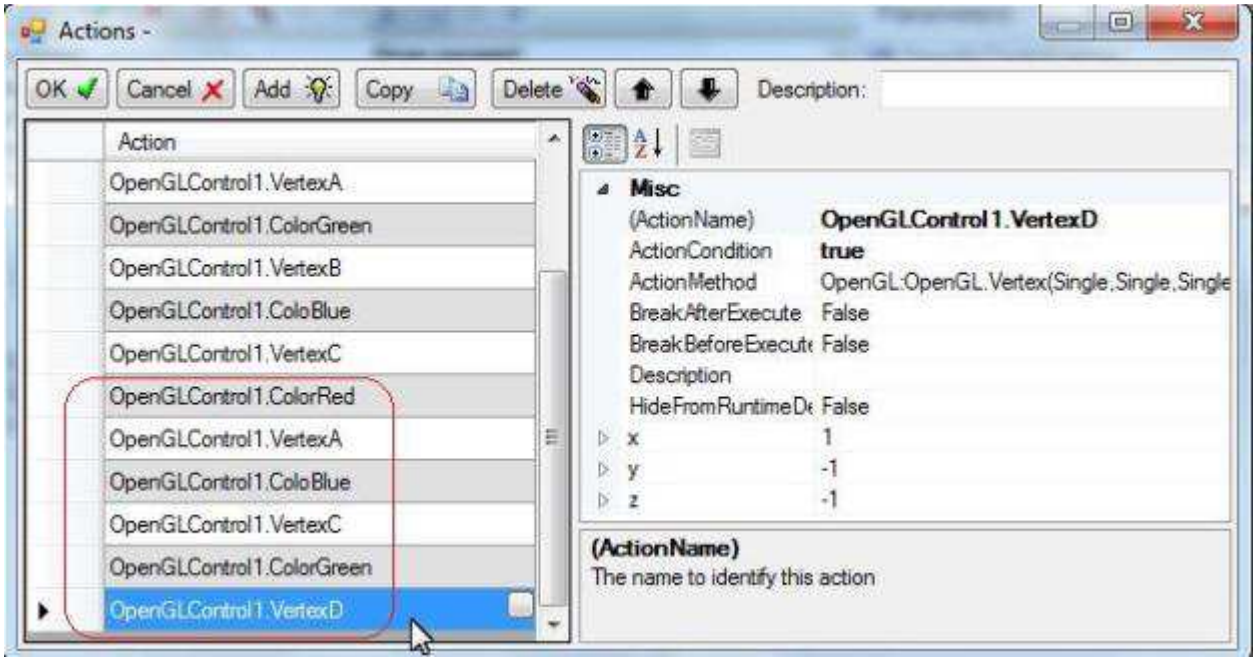
Add the two actions, OpenGLControl1.ColorBlue and OpenGLControl1.VertexC, we already created, to the list:



Draw vertex D with green

Add existing OpenGLControl1.ColorGreen to the list.

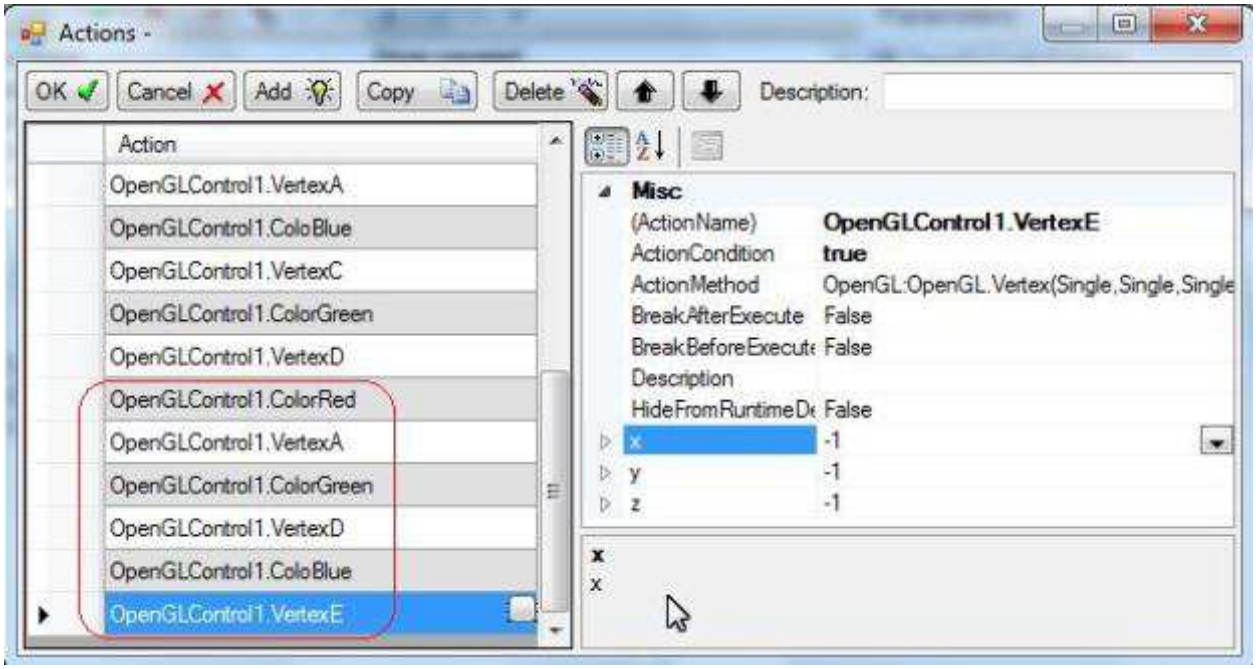
Copy OpenGLControl1.VertexC to make a new OpenGLControl1.VertexD:



We thus finish drawing triangle ACD.

Draw triangle ADE

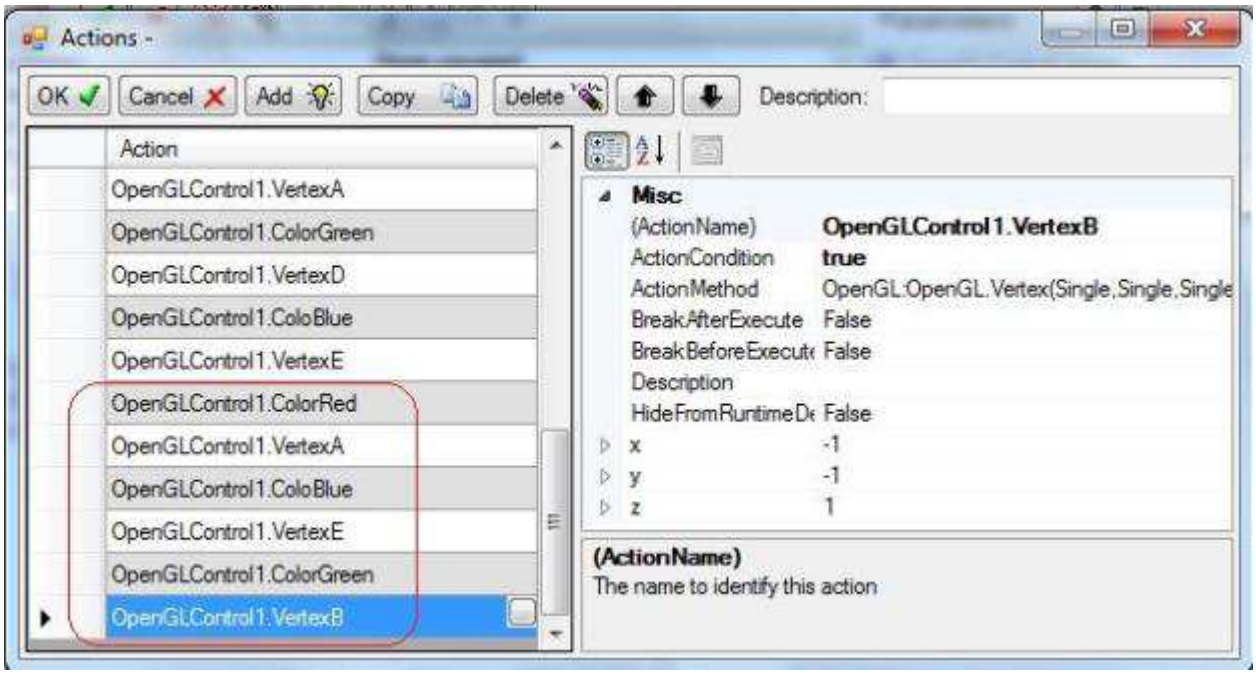
We add another 6 actions to draw vertex A, D, E:



Where action OpenGLControl1.VertexE is a new action created by copy an existing vertex action and apply the coordinates of point D (-1, -1, -1).

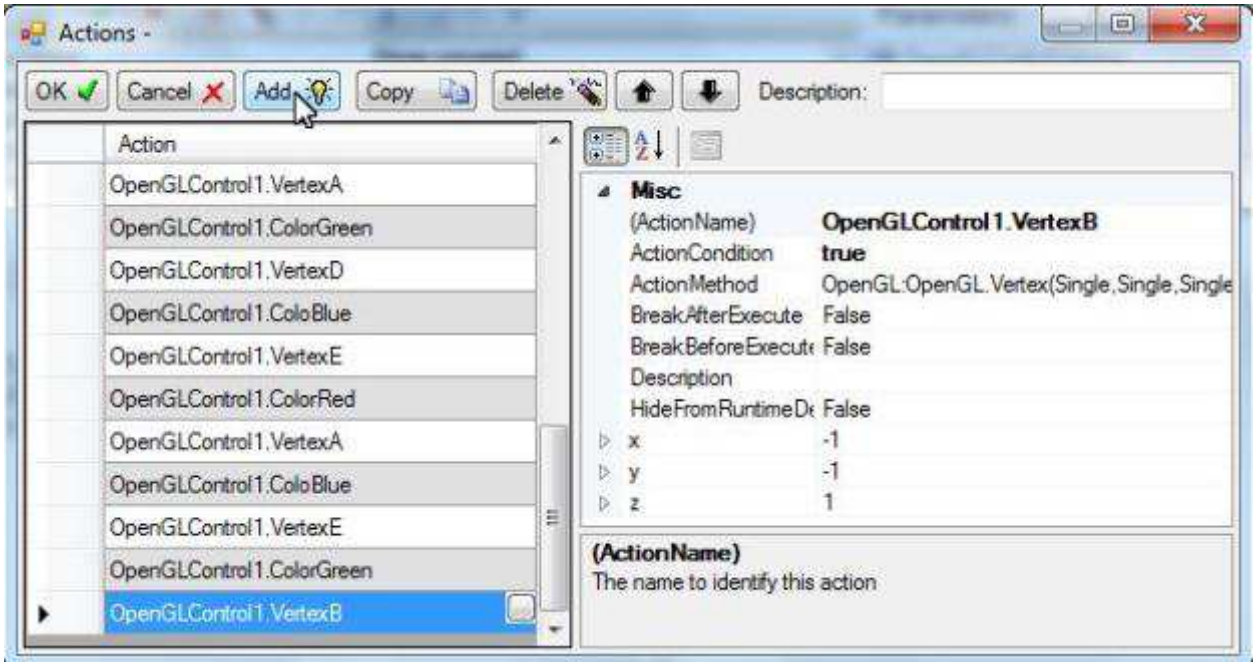
Draw triangle AEB

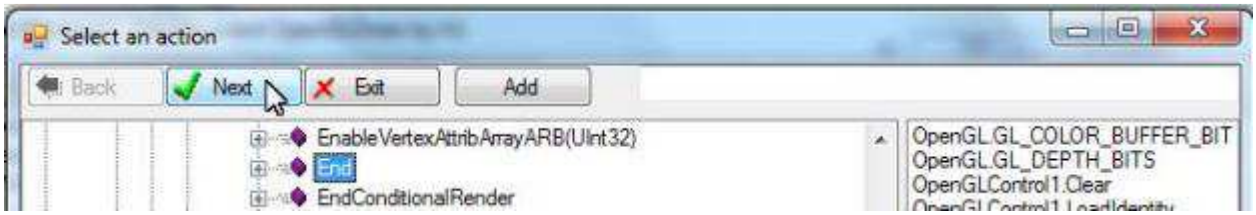
All 6 actions needed are already created. Add them to the list:

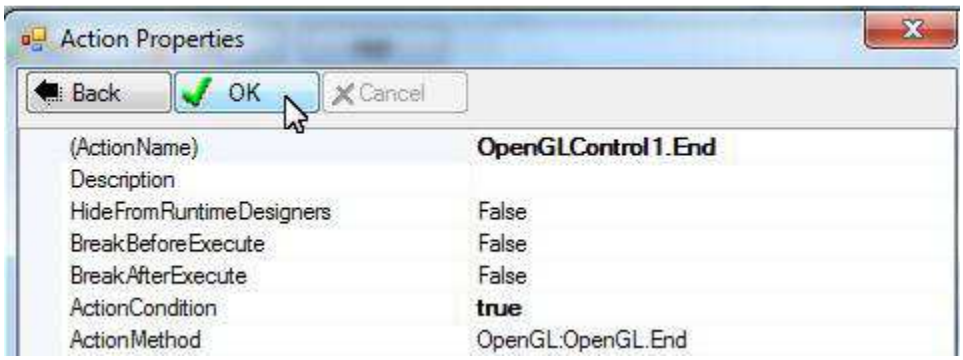


End of drawing triangles

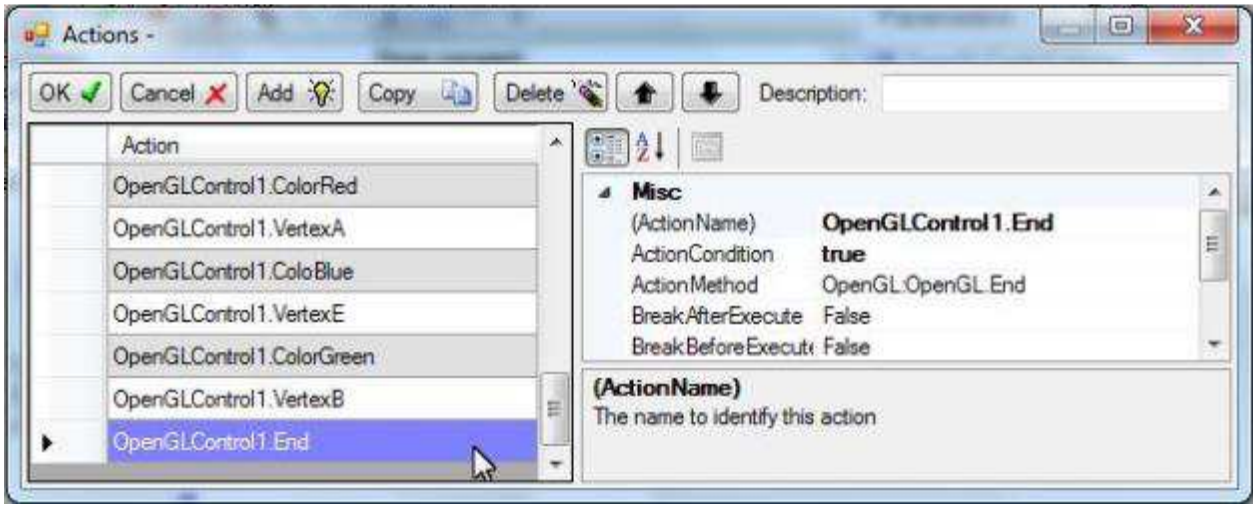
Use an End action to finish drawing triangles because we finish drawing all triangles for the pyramid.





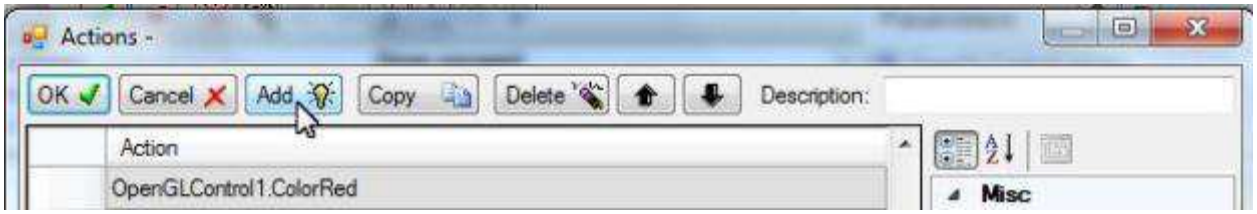


The End action appears at the end of the list:

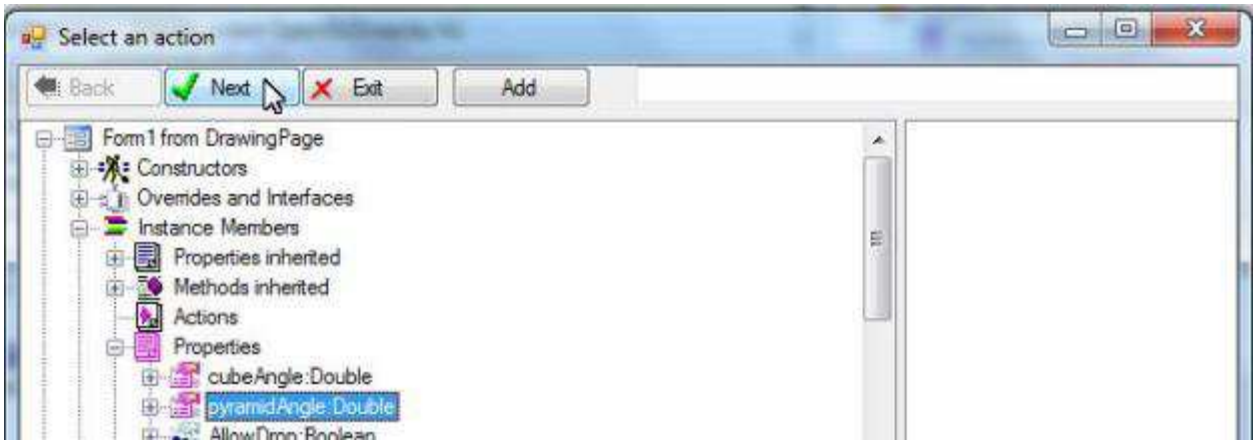


Increase pyramid rotation angle

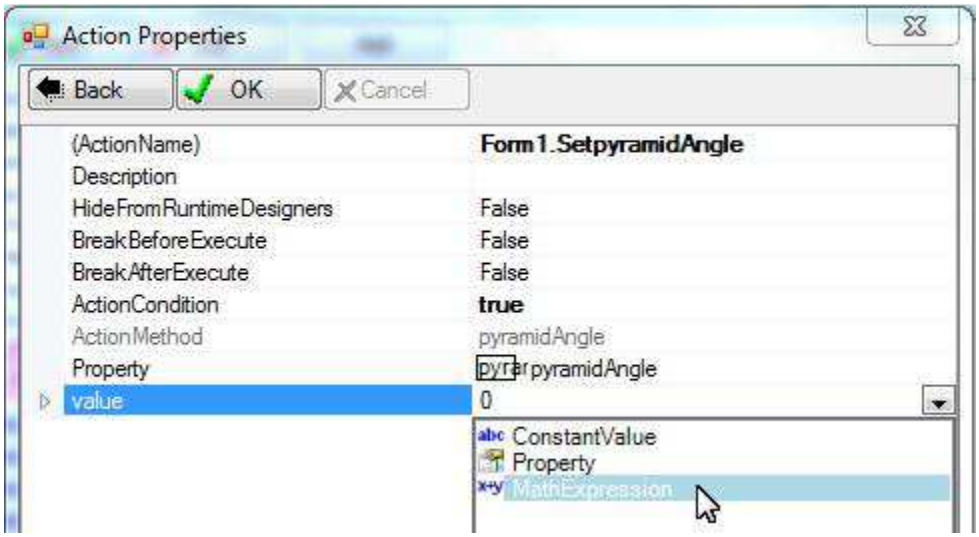
We increase the property, pyramidAngle, for the next frame. Click Add:

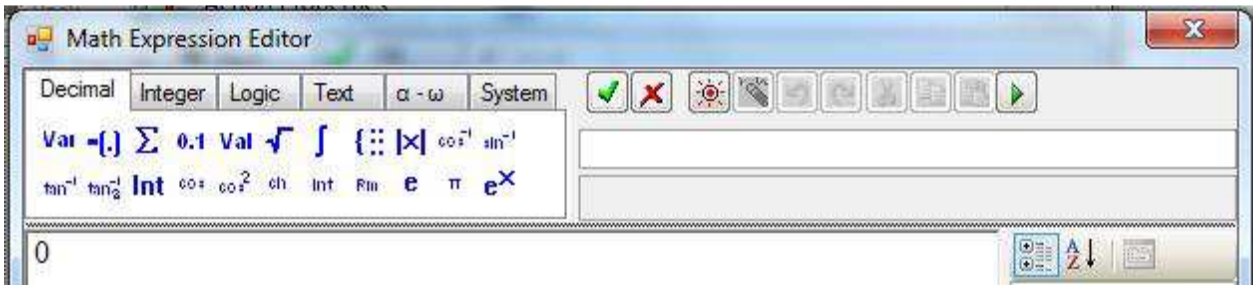


Select property pyramidAngle and click Next:

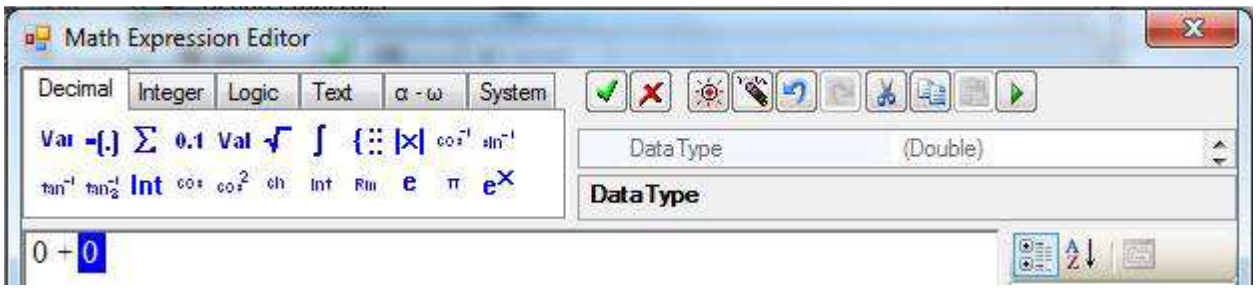


Select Math Expression to calculate the next pyramid rotation angle:

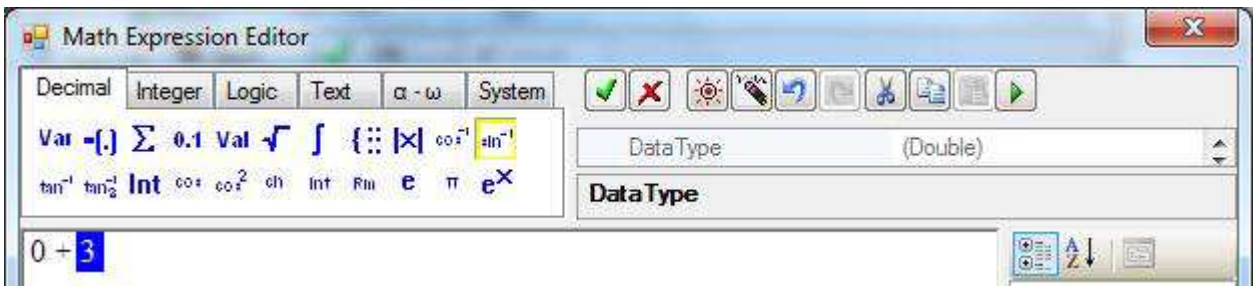




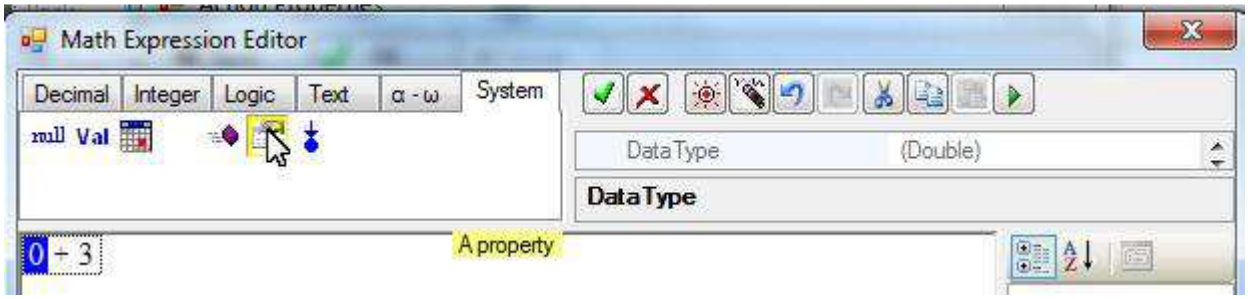
We want to add 3 degree to the existing pyramid angle. So, press "+" key to get a plus expression:

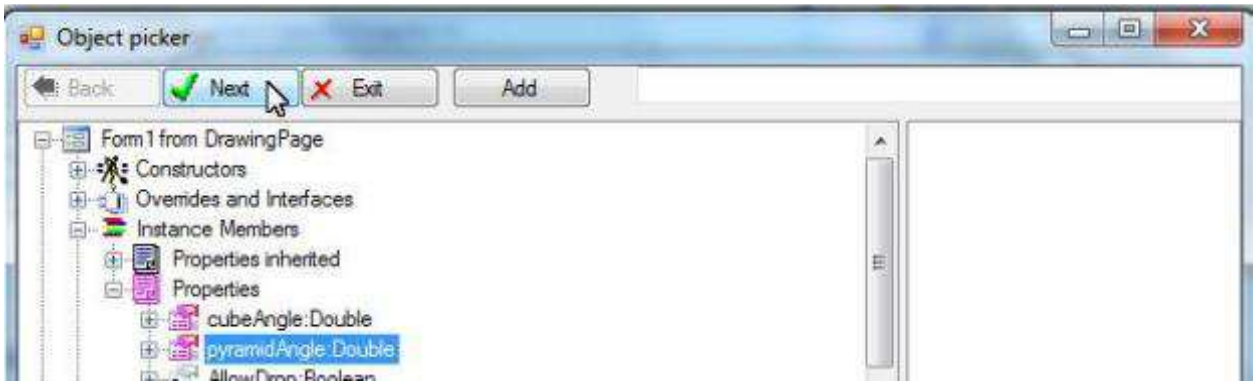


Press 3 for adding 3 degrees:



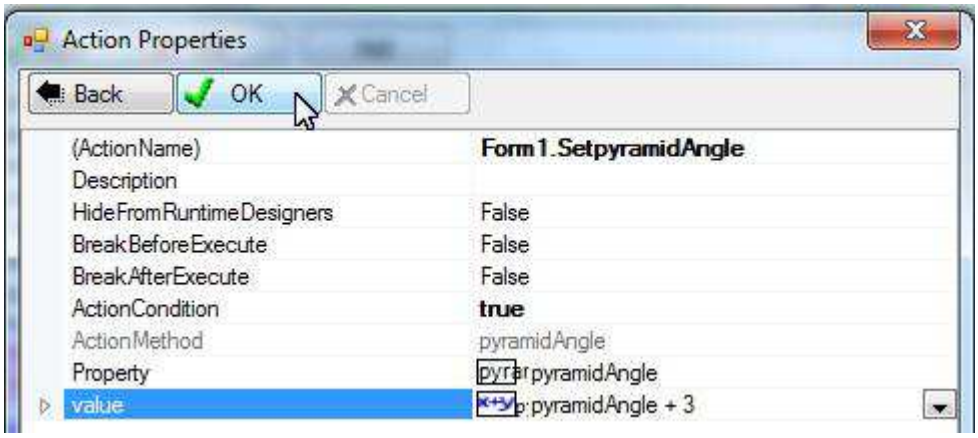
Select the first item and click the Property icon to select the existing pyramid angle:



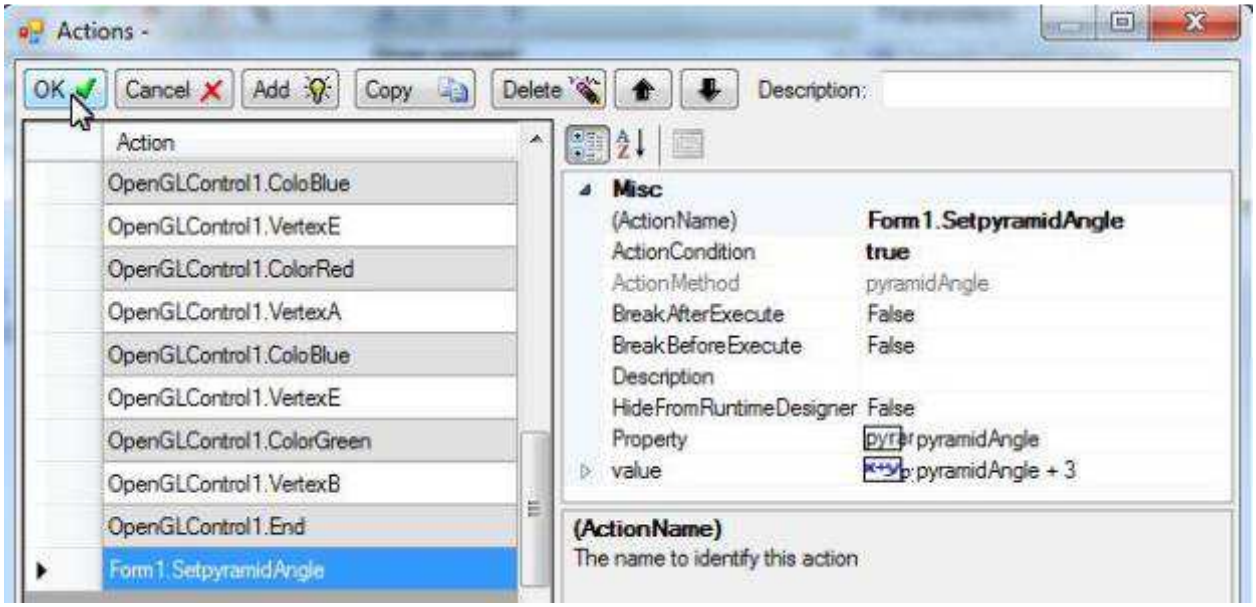


This is the calculation for the next pyramid rotation angle:

Click OK to finish creating this action:



These are all the actions for drawing the pyramid.

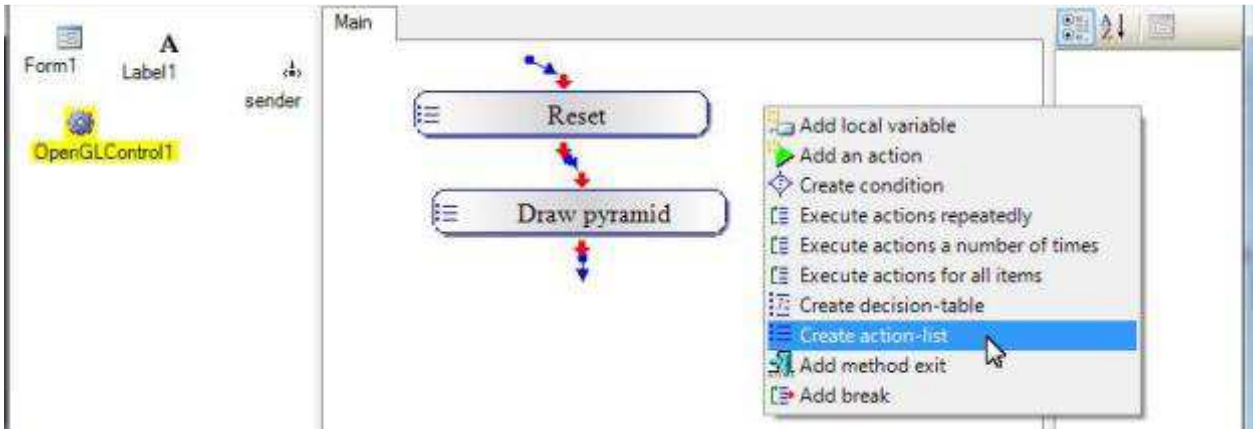


Draw Cube

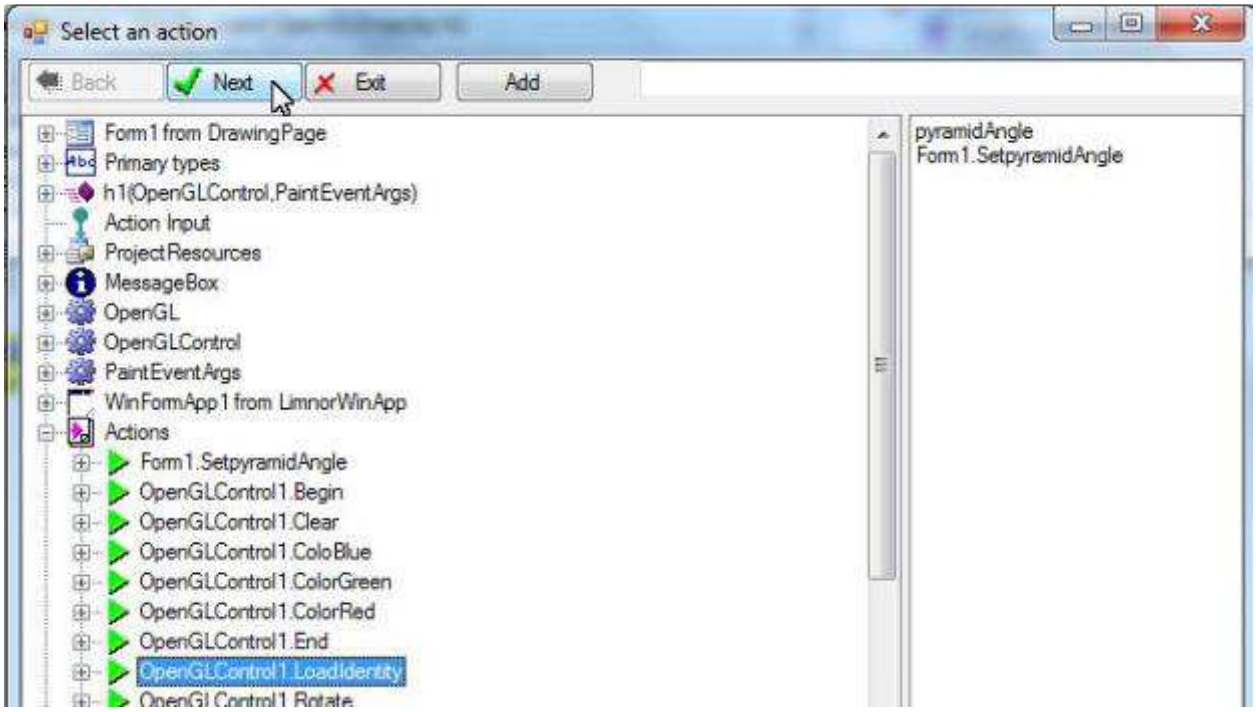
Prepare for drawing cube – reset coordinate transformations

A LoadIdentity action is needed to reset coordinate transformations. We already created it for drawing pyramid. We may reuse that action here.

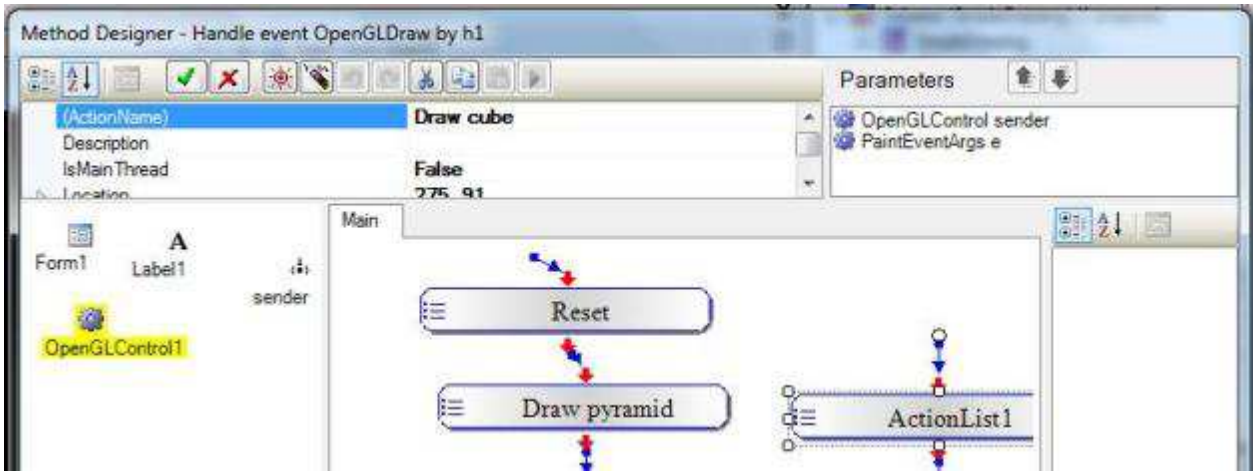
We create an action list and name it Draw Cube to include all cube-drawing actions:



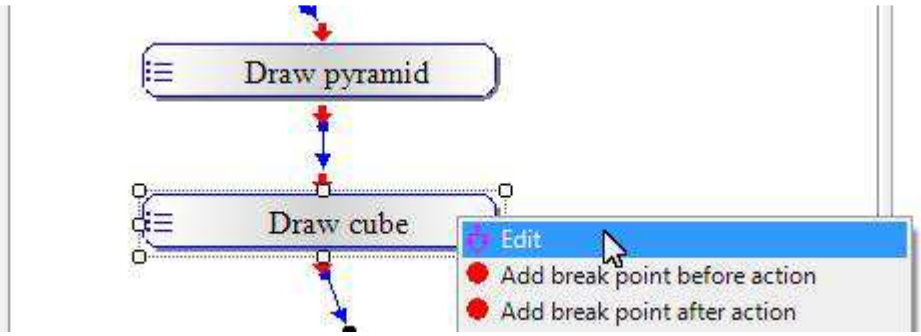
Select the LoadIdentity action we created before and click Next



A new action list is created. Name it “Draw cube”:



Link it to the last action list and edit it to add other cube-drawing actions to it:



Prepare for drawing cube – set location

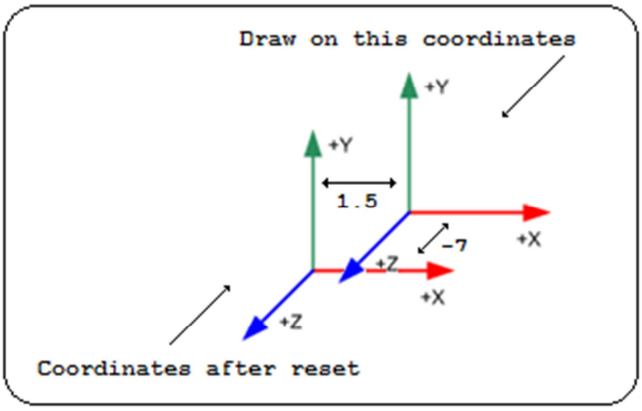
We want to draw the cube in the right side of the window.

So, we first use a Translation action to move the drawing location to the right side of the window.

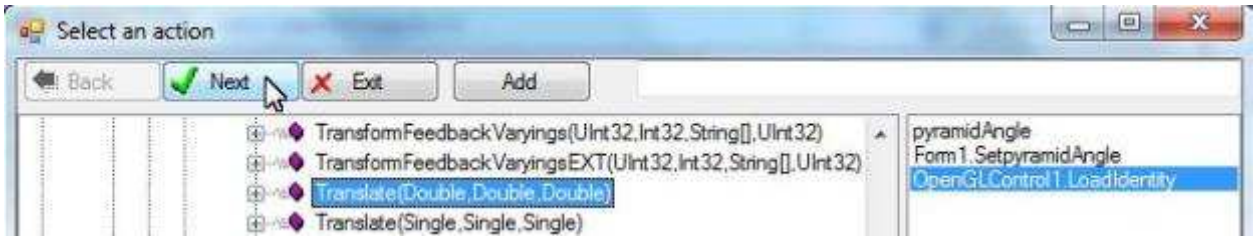
We do it by moving the x-axis by 1.5.

We also move z-axis by -7 to make the cube "inside" into the window to create a depth.

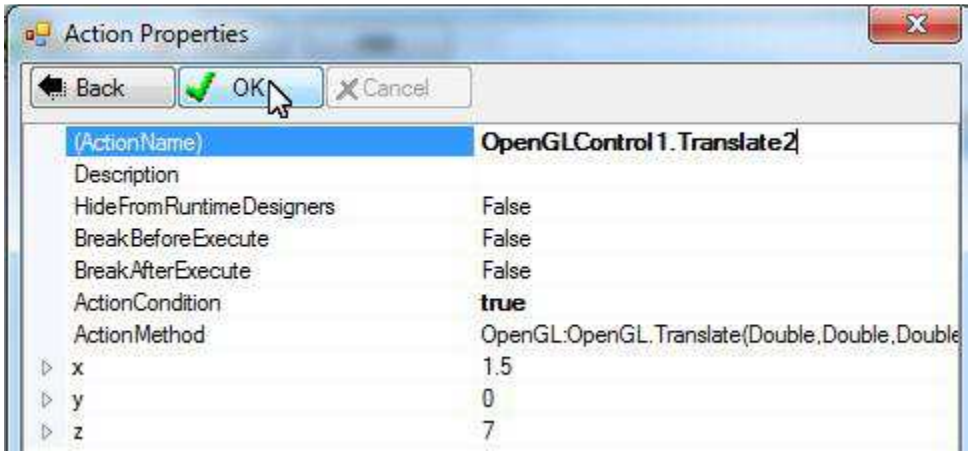
The effects can be illustrated by the following figure:



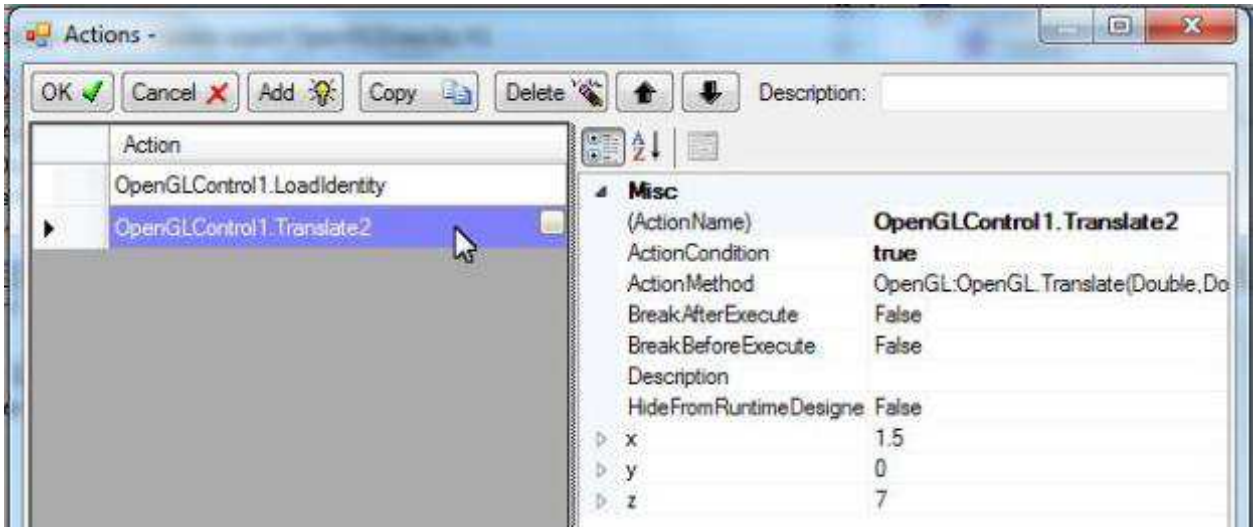
Let's add a Translation action to do it:



Set x to 1.5 and z to -7:



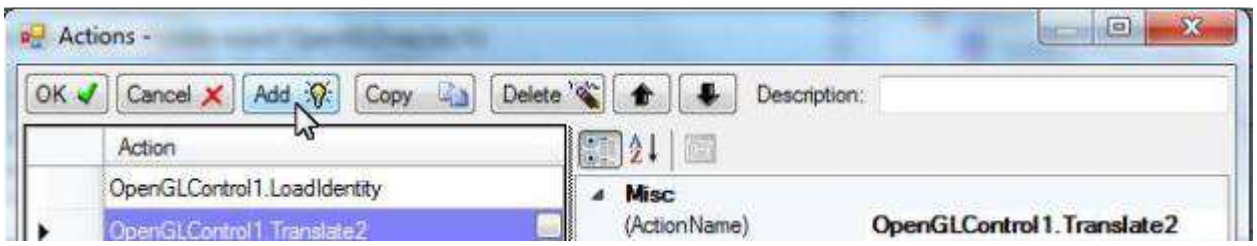
The action appears in the list:

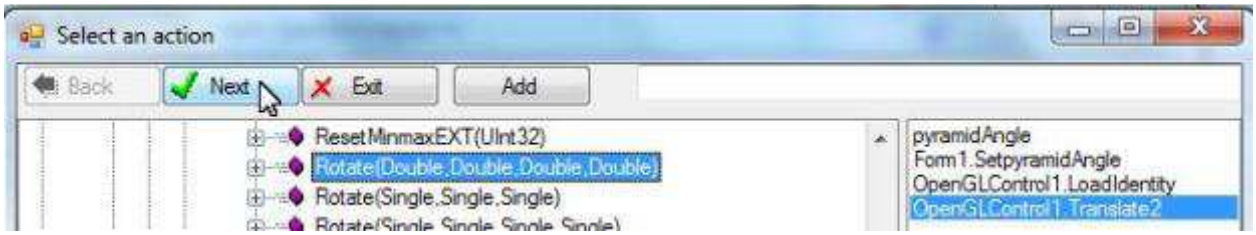


Prepare for drawing cube – rotate

We are drawing one frame in handling one OpenGLDraw event. We need to rotate it using the property cubeAngle. After drawing the frame, we will increase the rotate angle so that each frame will use a different angle to create a rotating effect.

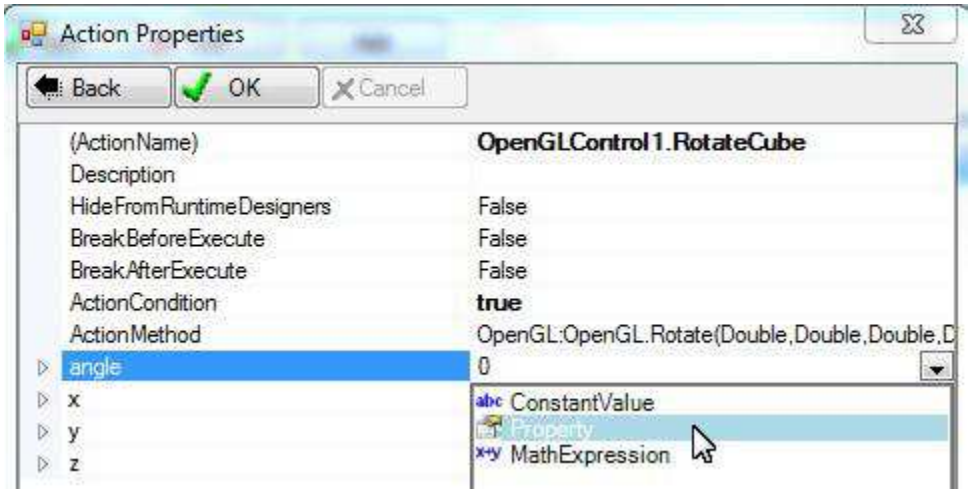
Let's create a Rotate action to rotate the coordinate.

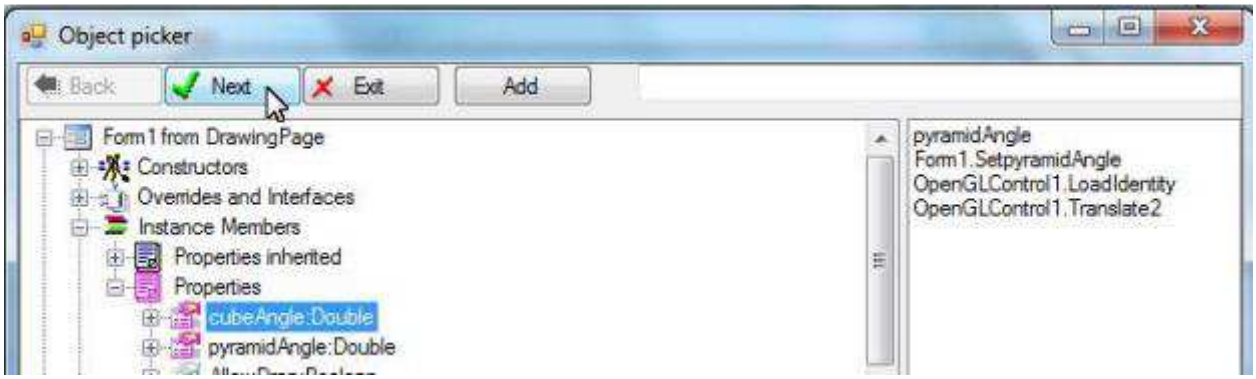




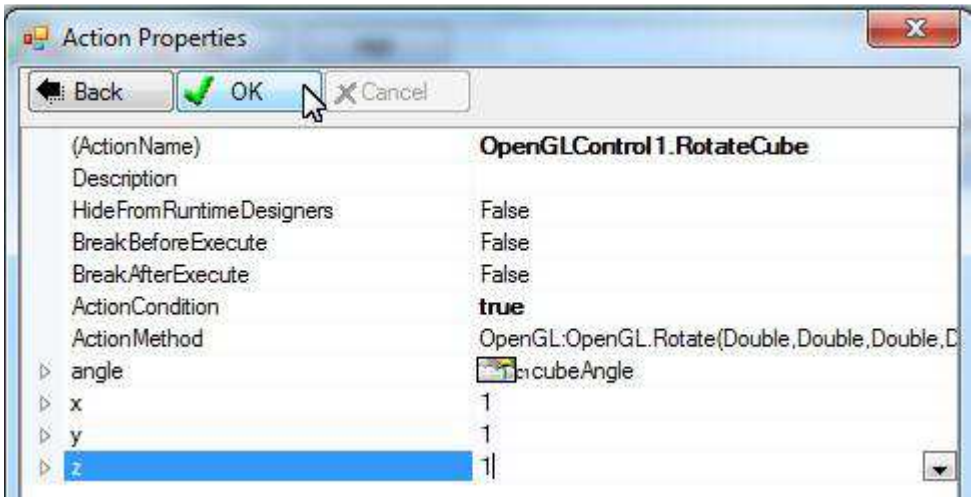
Name the action RotateCube:

Select Property for “angle” to use cubeAngle property:

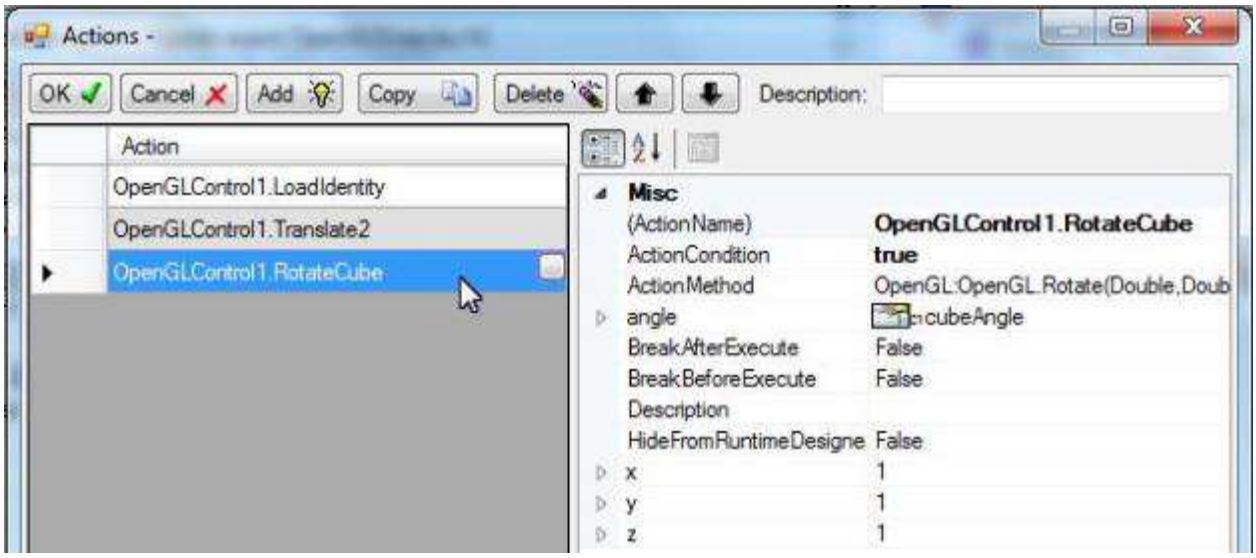




Set 1 to x, y, and z to rotate on all axes:

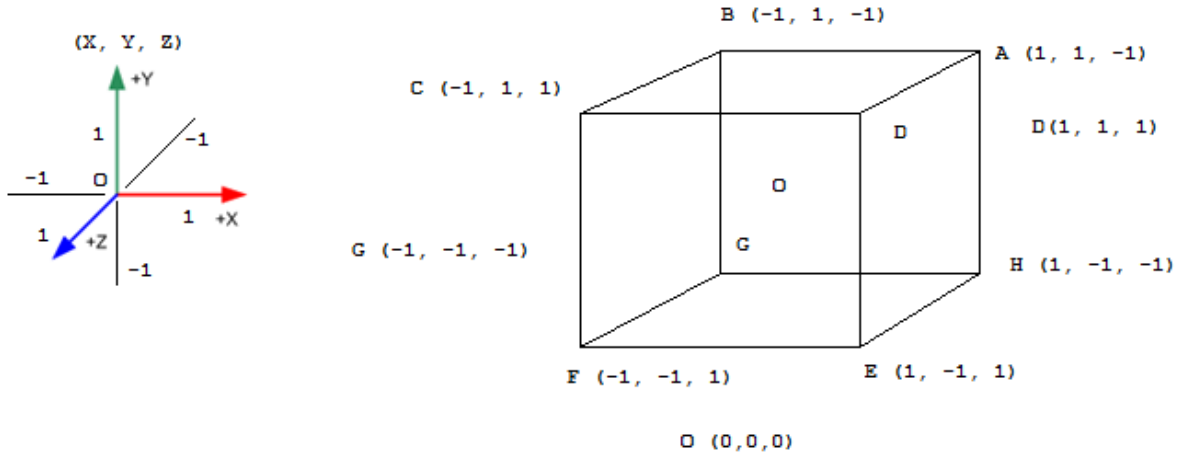


The action appears in the list:



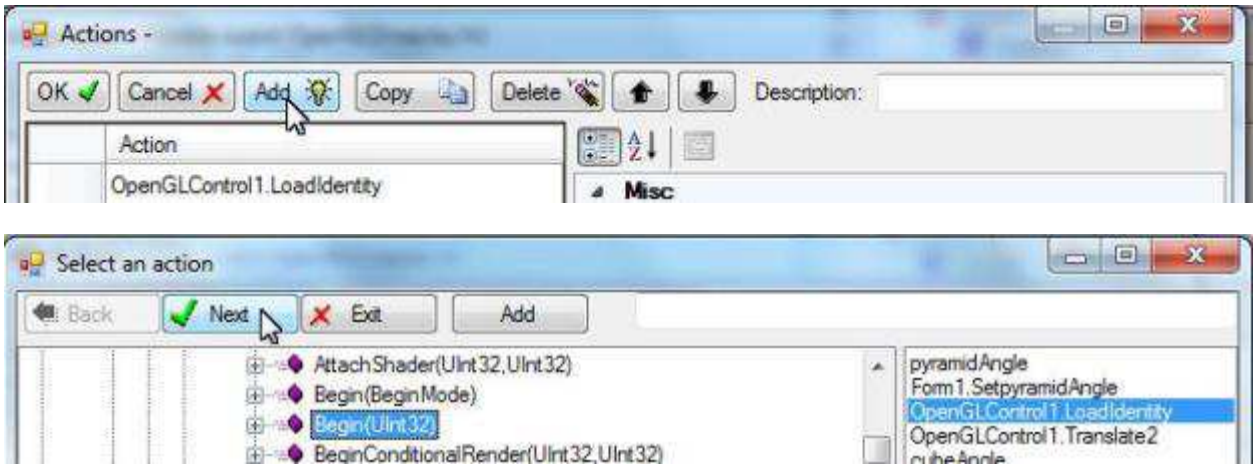
Drawing cube sides

A cube has 6 sides:

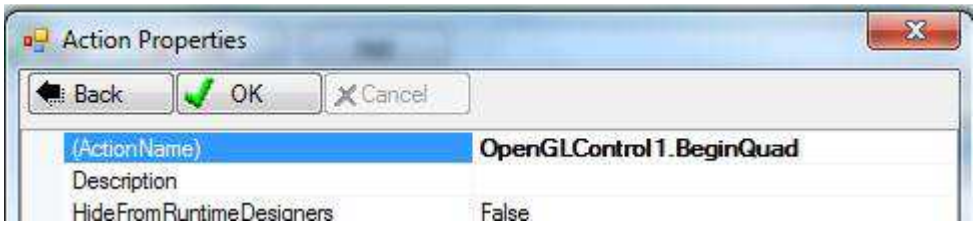


Begin drawing quadrilateral

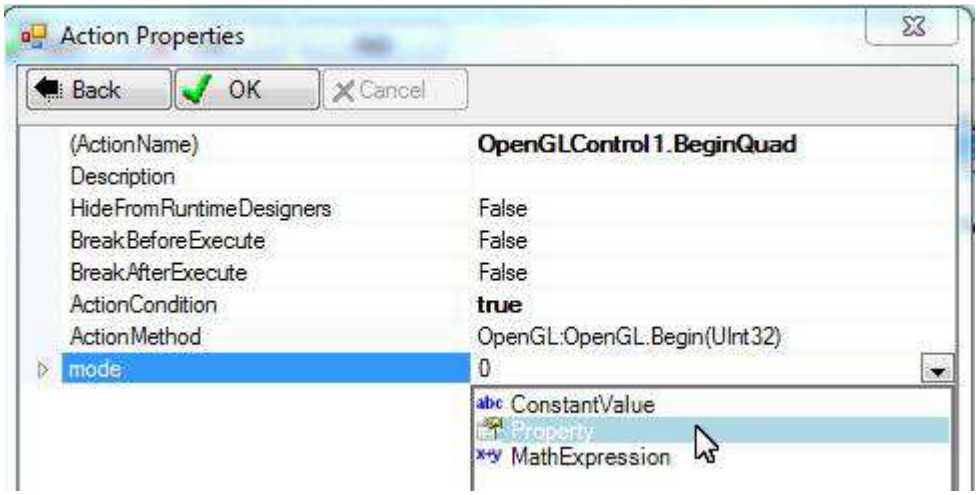
Use a Begin action to start drawing quadrilaterals.



Name it BeginQuad:



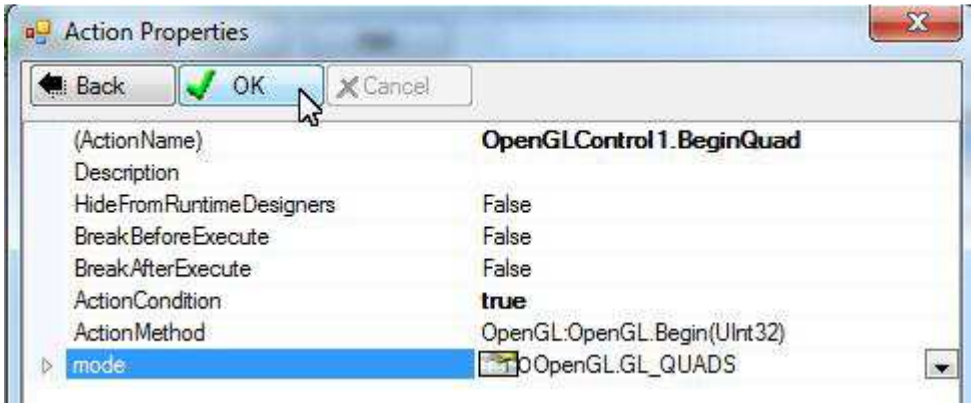
Select Property for “mode” to select a proper mode from OpenGL:



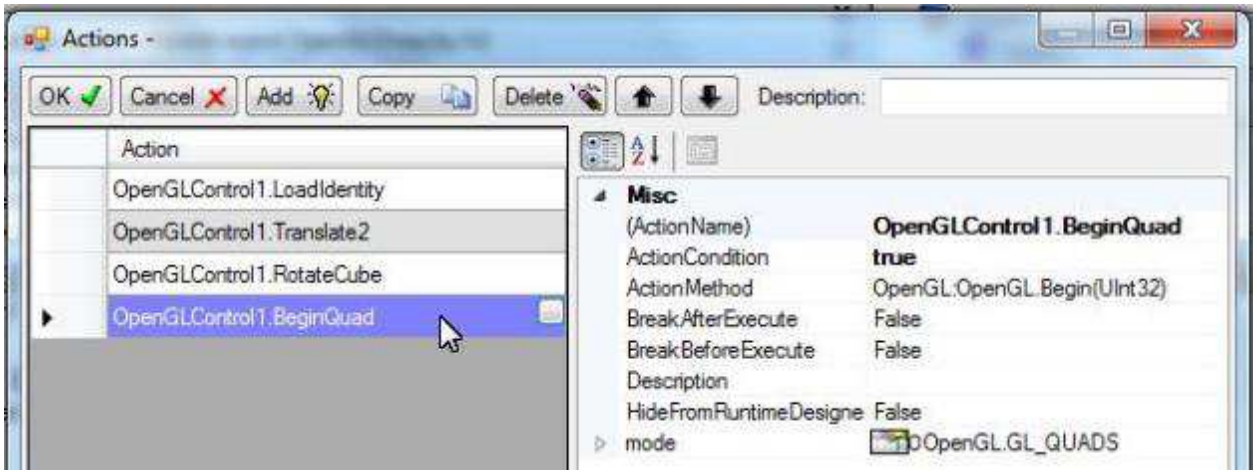
Find the mode definitions in “Static fields” under OpenGL:

Select GL_QUADS for drawing quadrilaterals:

Click OK:

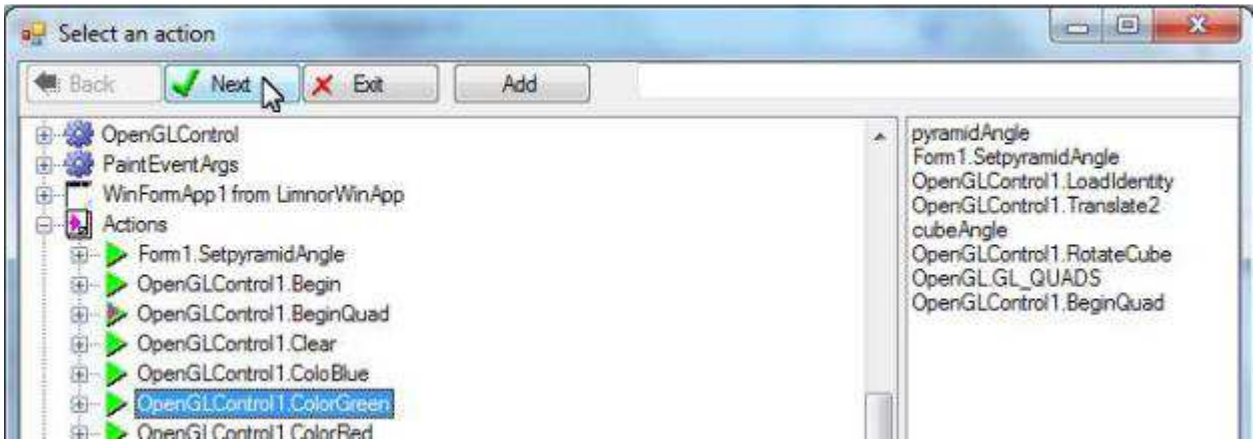


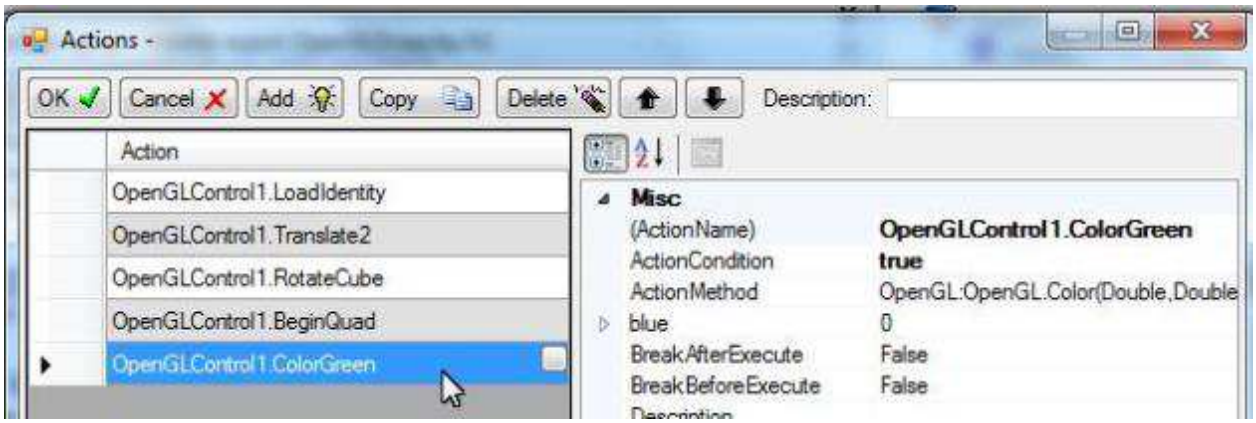
The action appears in the list:



Draw cube side ABCD with color green

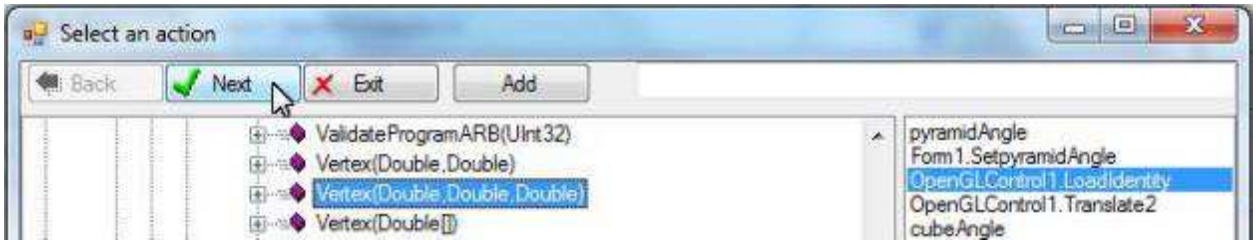
Add ColorGreen action we already created:





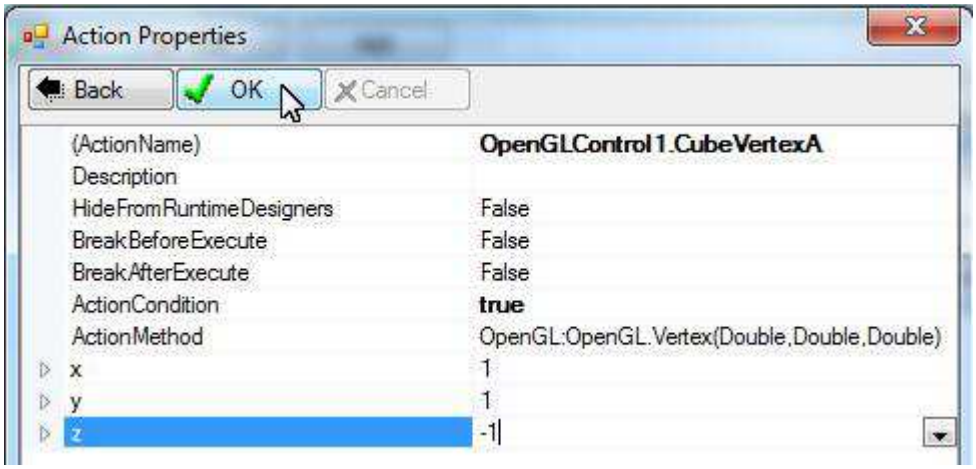
Create a Vertex action to draw vertex A:

Select Vertex method:

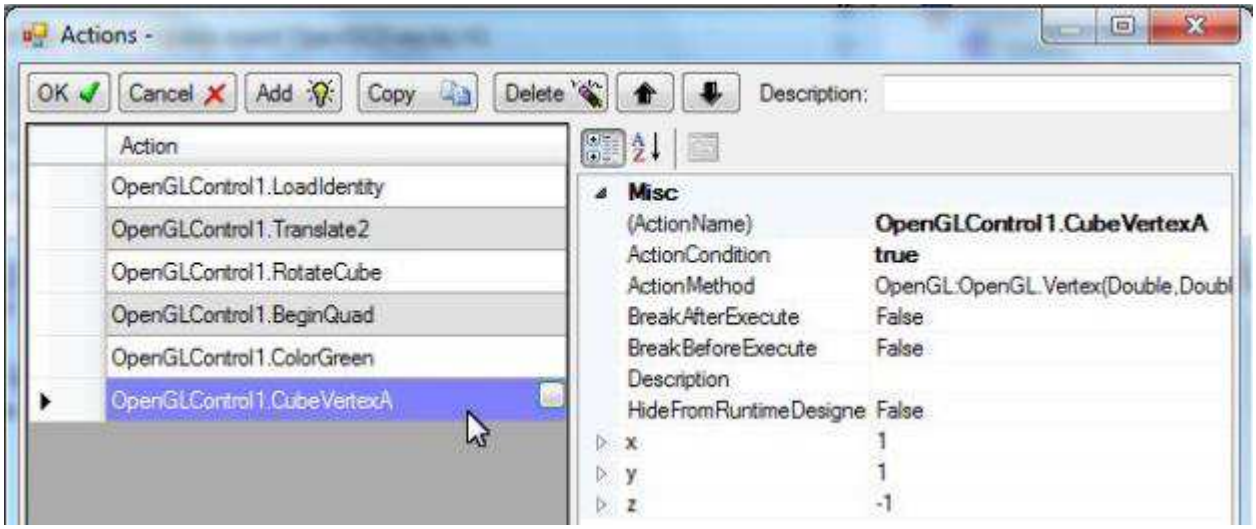


Name the action CubeVertexA:

Apply the coordinates of point A to the action:



The action appears in the list:



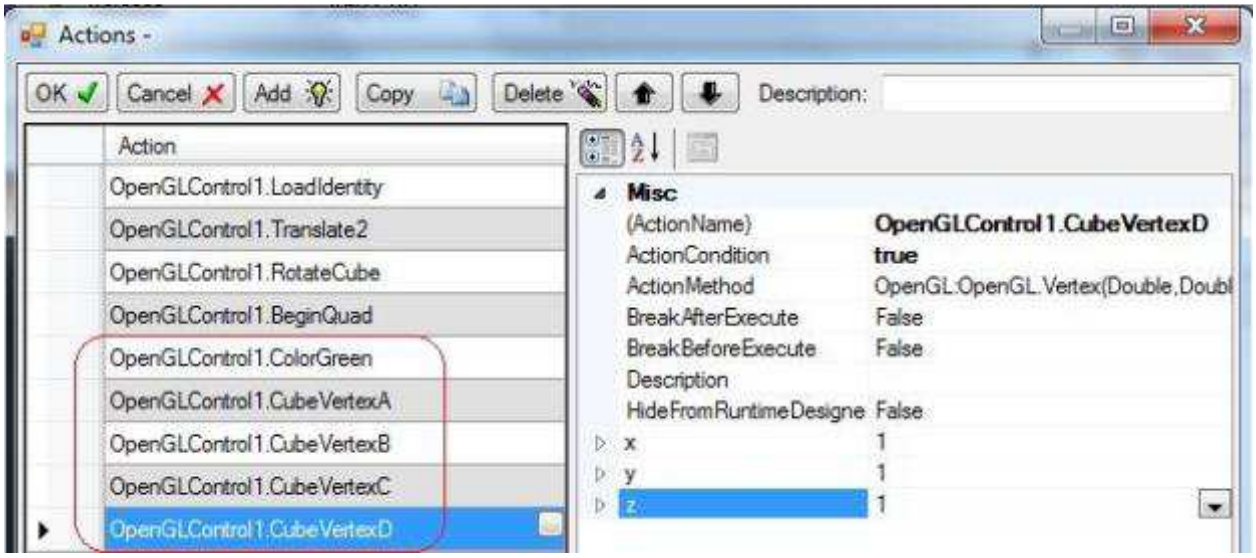
By copying action CubeVertexA, we add actions for point B, C, and D, using their respective coordinates:

B: -1, 1, -1

C: -1, 1, 1

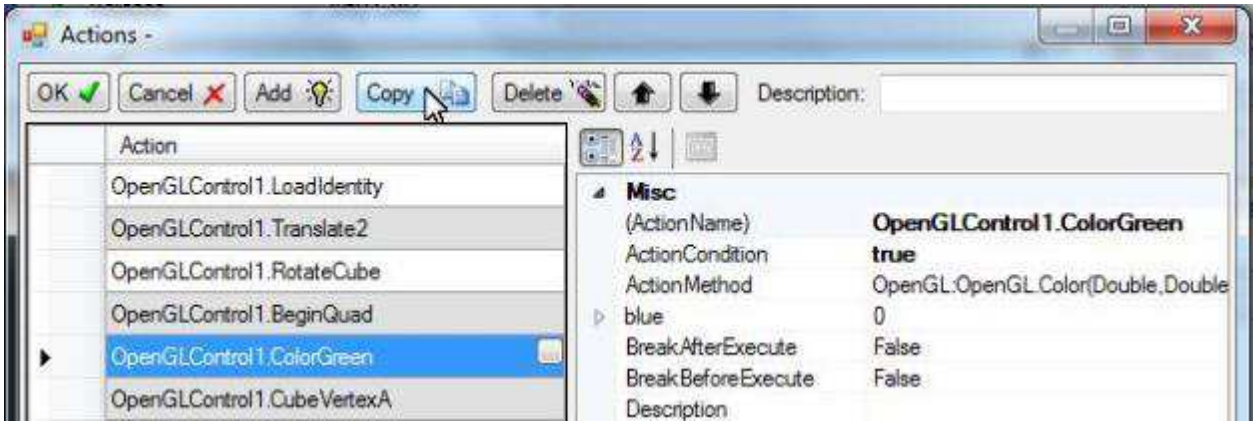
D: 1, 1, 1

These 5 actions draw side ABCD for the cube:

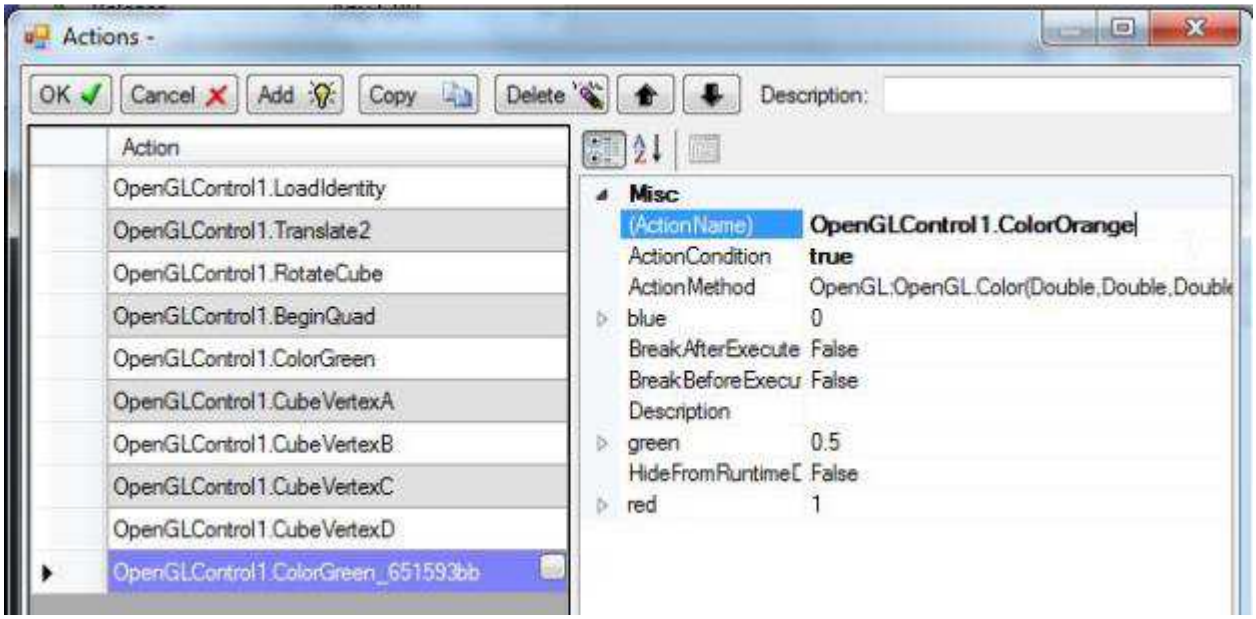


Draw cube side EFGH with color orange

Copy action ColorGreen to add a new action:



Set “red” to 1, set “green” to 0.5, set “blue” to 0, and name it ColorOrange:



Copy and create Vertex action for point E, F, G, and H, using each point's coordinates:

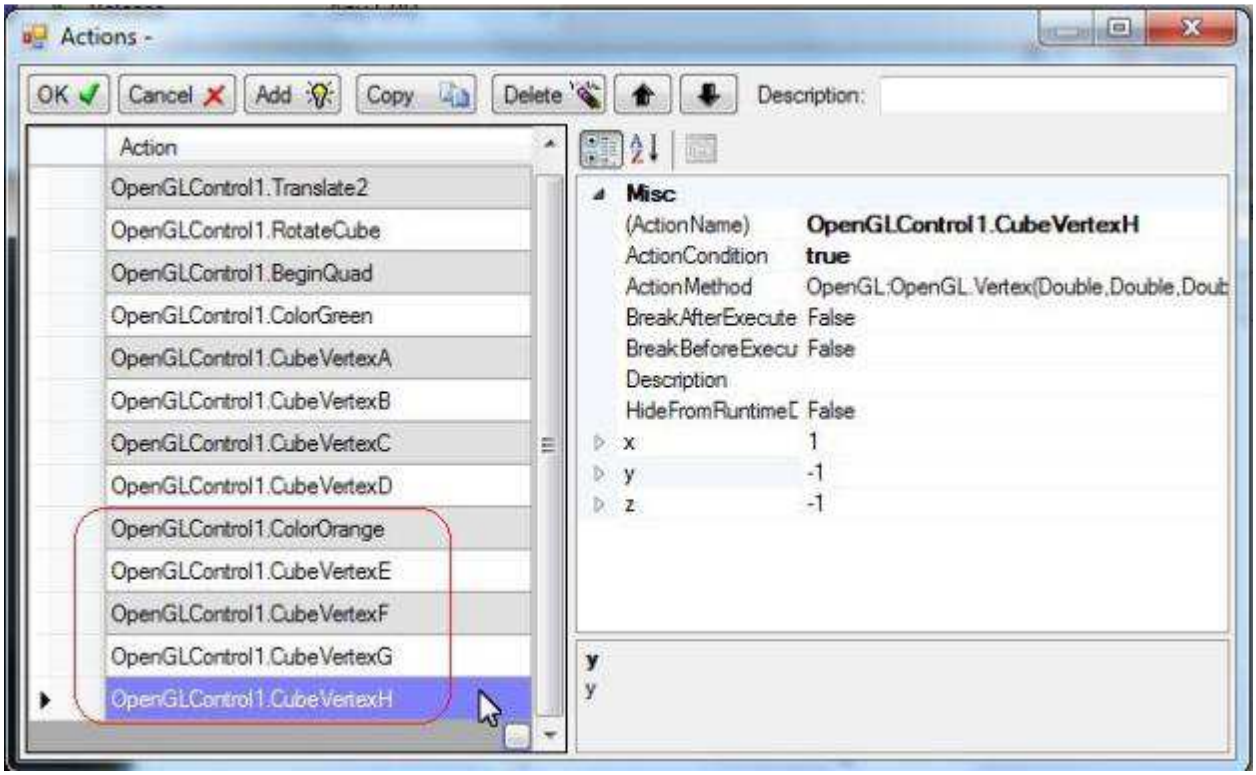
E: 1, -1, 1

F: -1, -1, 1

G: -1, -1, -1

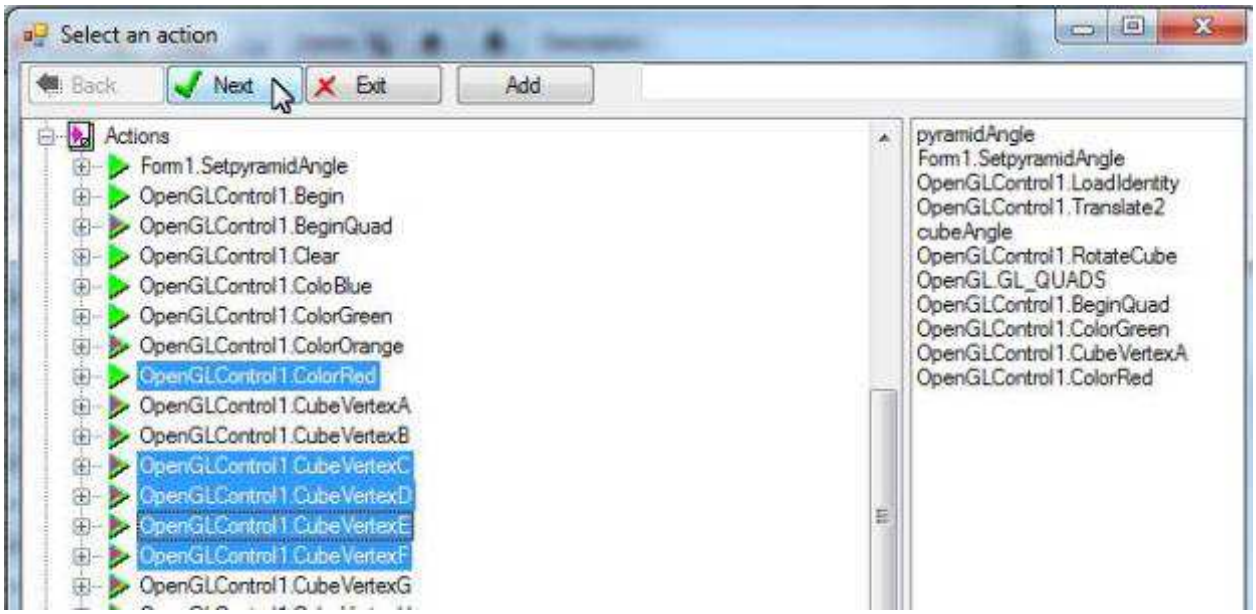
H: 1, -1, -1

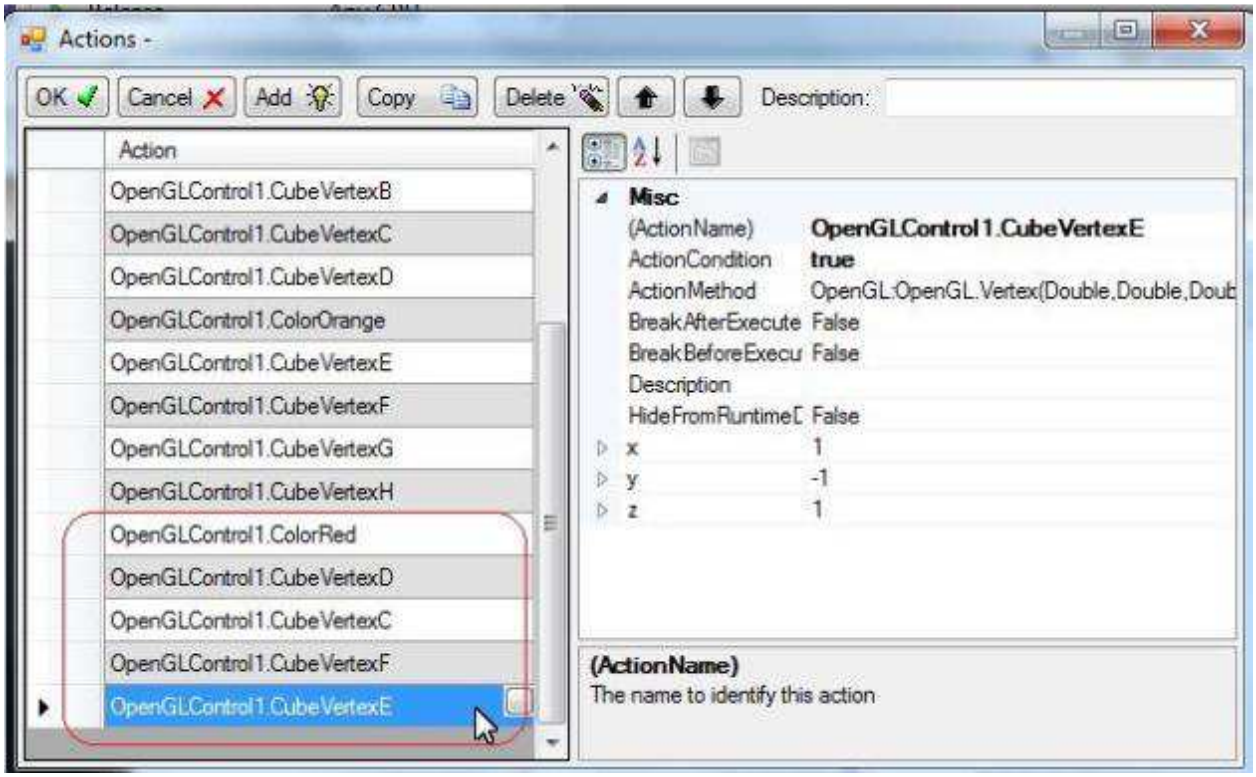
These 5 actions draw side EFGH:



Draw cube side DCFE with color red

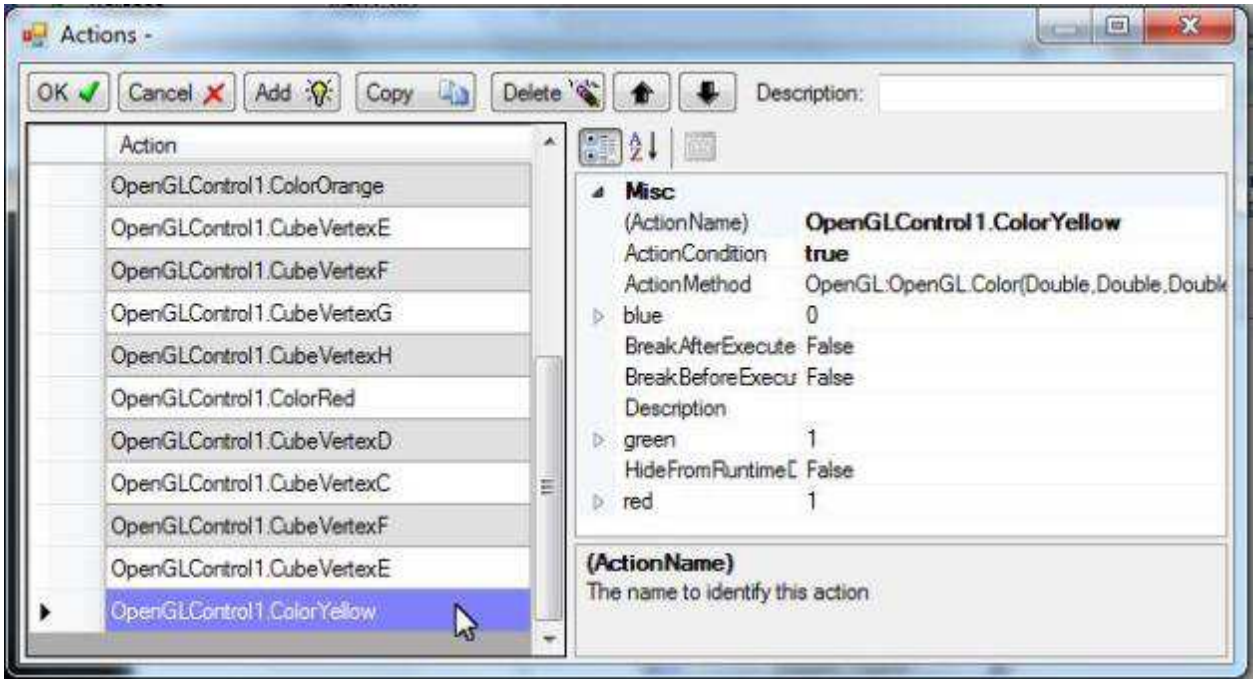
All actions needed are already created. Add them to the list:





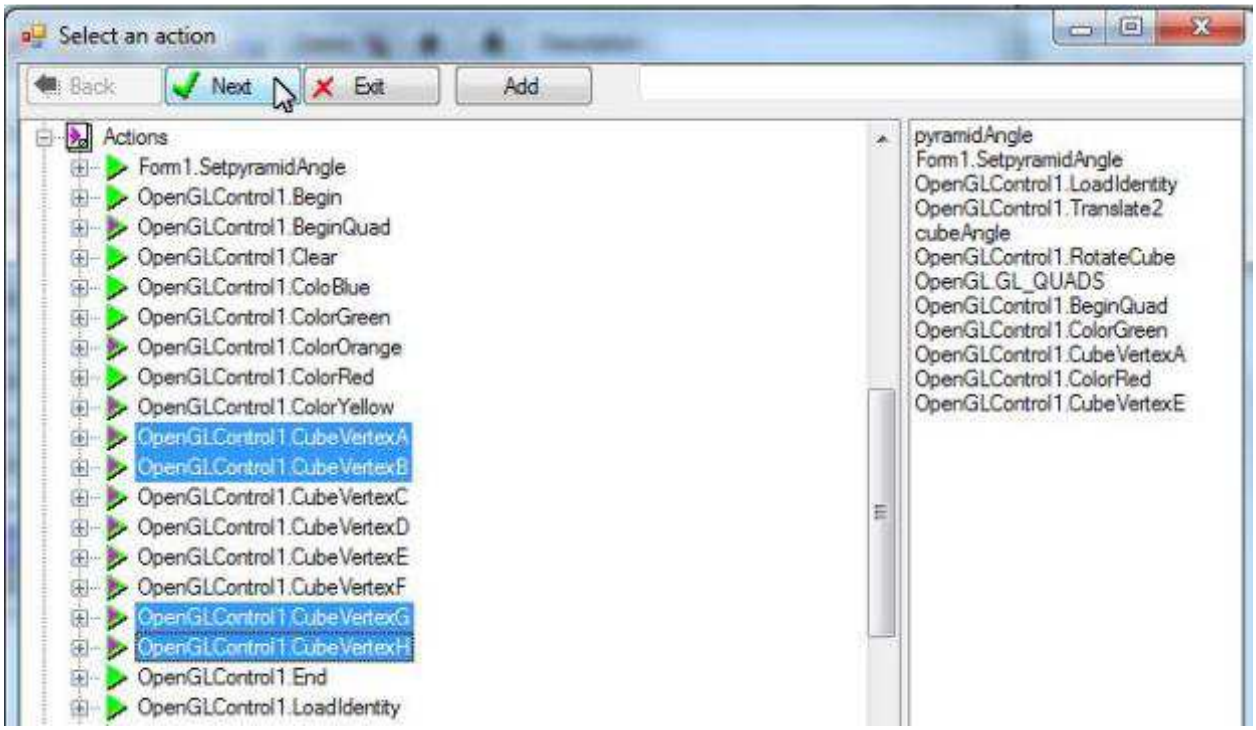
Draw cube side HGBA with color yellow

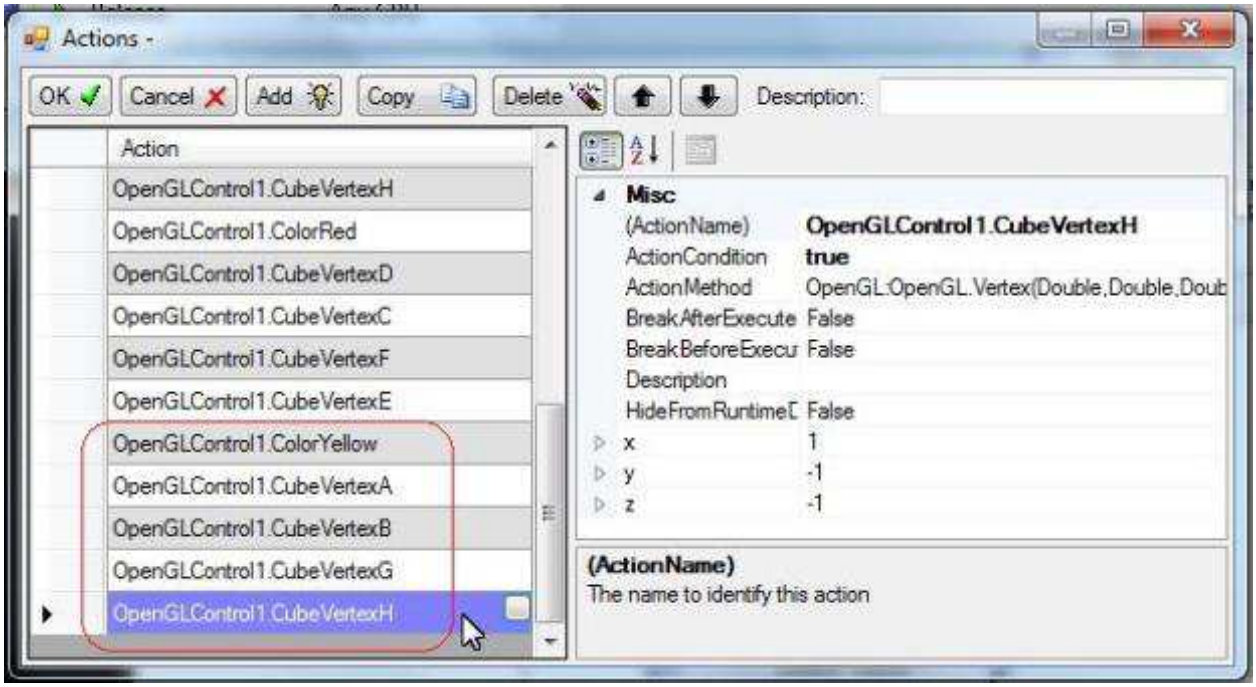
Copy and create a Color action for color yellow (blue=0, green=1, red=1):



All actions for vertex points needed are already available. Add them to the list:

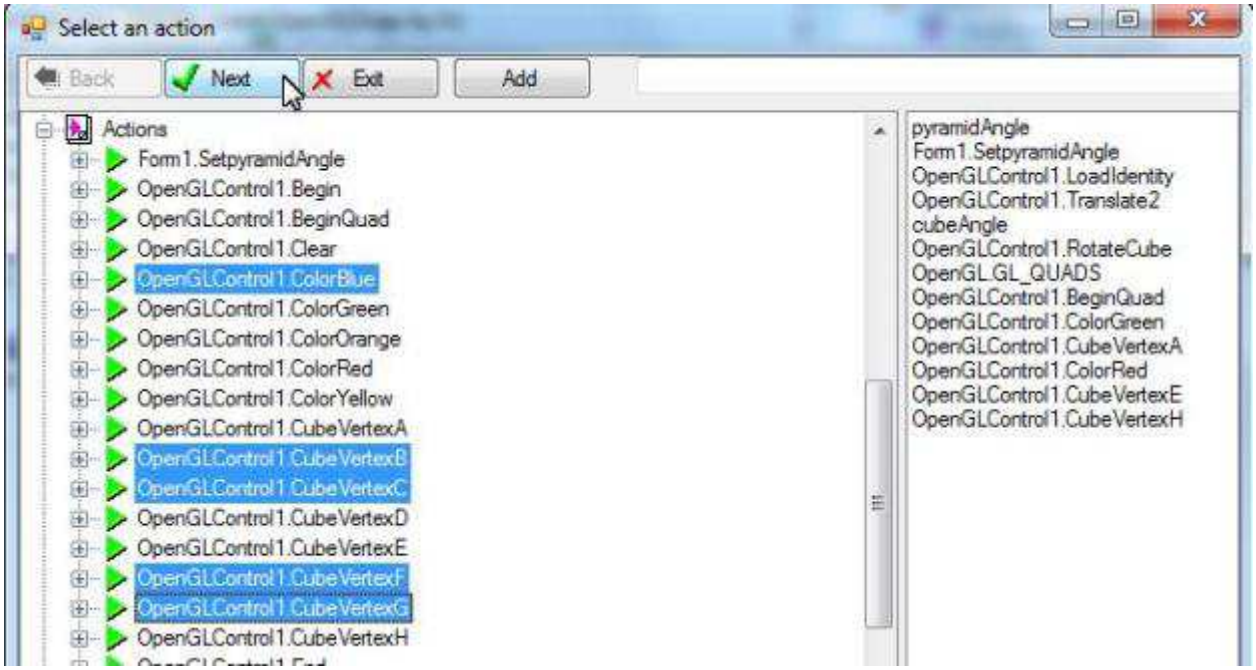

Actions -
OK
Cancel
Add
Copy
Delete
Description:
Action

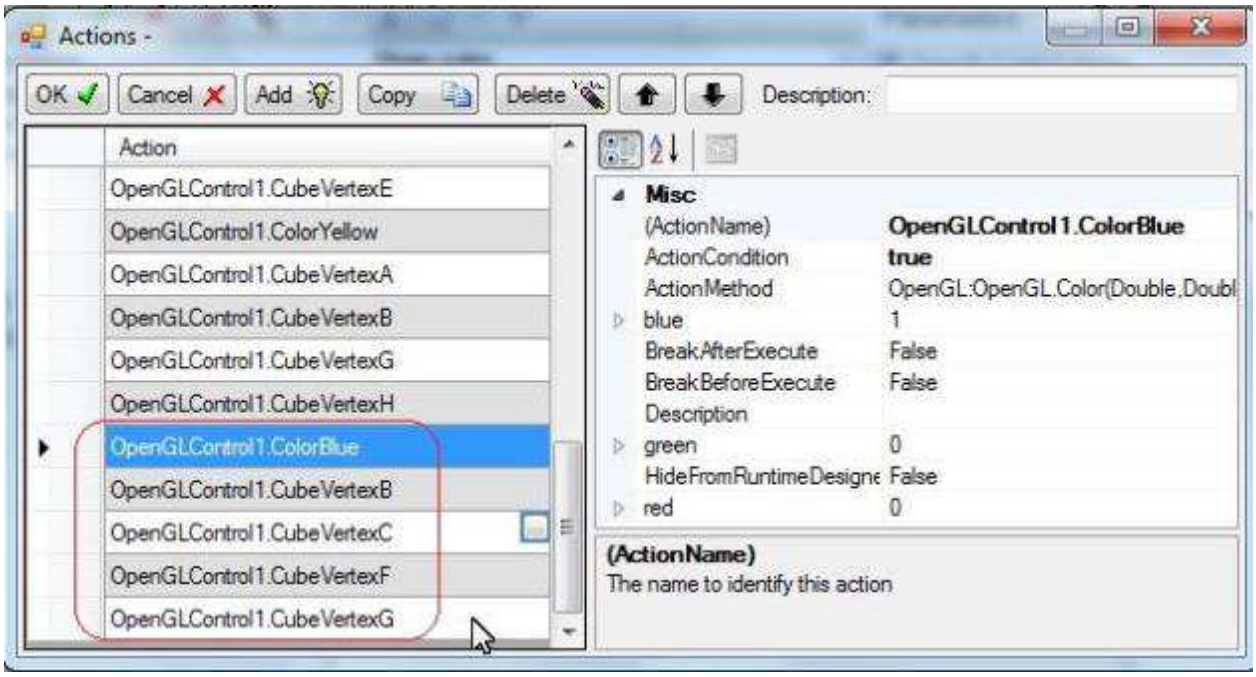

Select an action
Back
Next
Exit
Add
Actions
Form1.SetpyramidAngle
OpenGLControl1.Begin
OpenGLControl1.BeginQuad
OpenGLControl1.Clear
OpenGLControl1.ColoBlue
OpenGLControl1.ColorGreen
OpenGLControl1.ColorOrange
OpenGLControl1.ColorRed
OpenGLControl1.ColorYellow
OpenGLControl1.CubeVertexA
OpenGLControl1.CubeVertexB
OpenGLControl1.CubeVertexC
OpenGLControl1.CubeVertexD
OpenGLControl1.CubeVertexE
OpenGLControl1.CubeVertexF
OpenGLControl1.CubeVertexG
OpenGLControl1.CubeVertexH
OpenGLControl1.End
OpenGLControl1.LoadIdentity
pyramidAngle
Form1.SetpyramidAngle
OpenGLControl1.LoadIdentity
OpenGLControl1.Translate2
cubeAngle
OpenGLControl1.RotateCube
OpenGL.GL_QUADS
OpenGLControl1.BeginQuad
OpenGLControl1.ColorGreen
OpenGLControl1.CubeVertexA
OpenGLControl1.ColorRed
OpenGLControl1.CubeVertexE


Actions -
OK
Cancel
Add
Copy
Delete
Description:
Action
OpenGLControl1.CubeVertexH
OpenGLControl1.ColorRed
OpenGLControl1.CubeVertexD
OpenGLControl1.CubeVertexC
OpenGLControl1.CubeVertexF
OpenGLControl1.CubeVertexE
OpenGLControl1.ColorYellow
OpenGLControl1.CubeVertexA
OpenGLControl1.CubeVertexB
OpenGLControl1.CubeVertexG
OpenGLControl1.CubeVertexH
Misc
(ActionName) OpenGLControl1.CubeVertexH
ActionCondition true
ActionMethod OpenGL:OpenGL.Vertex(Double,Double,Doub
BreakAfterExecute False
BreakBeforeExecu False
Description
HideFromRuntimeC False
x 1
y -1
z -1
(ActionName)
The name to identify this action

Draw cube side CBGF with color blue

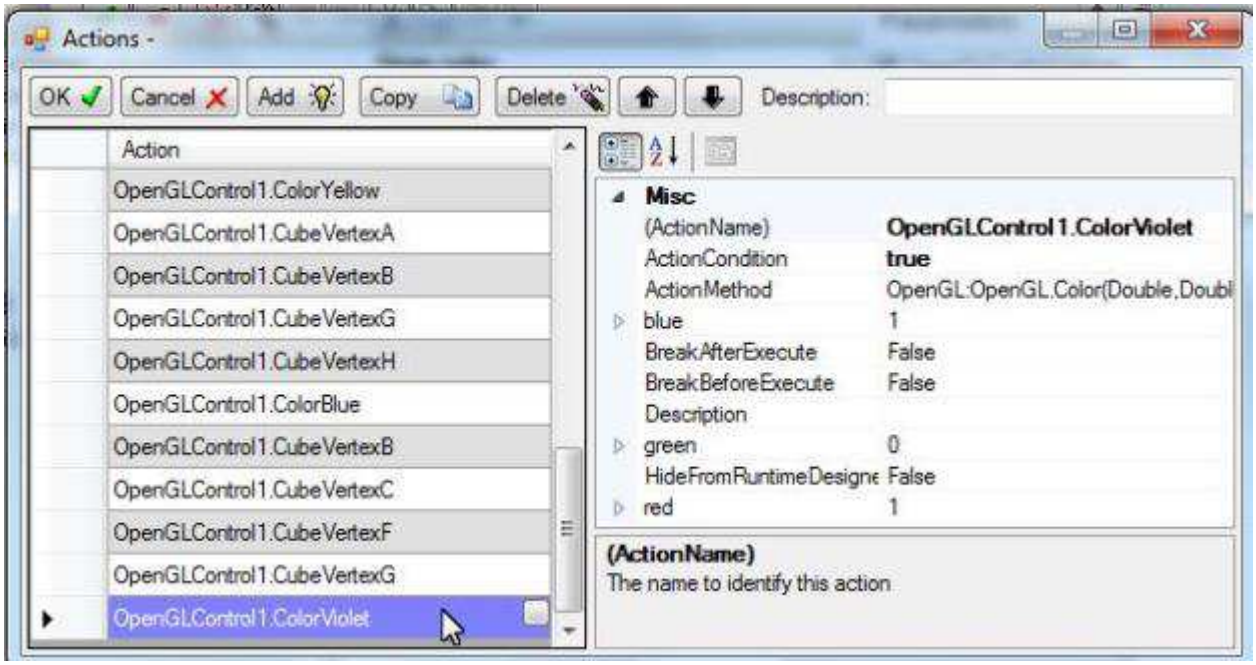
All actions are available. Add them to the list.



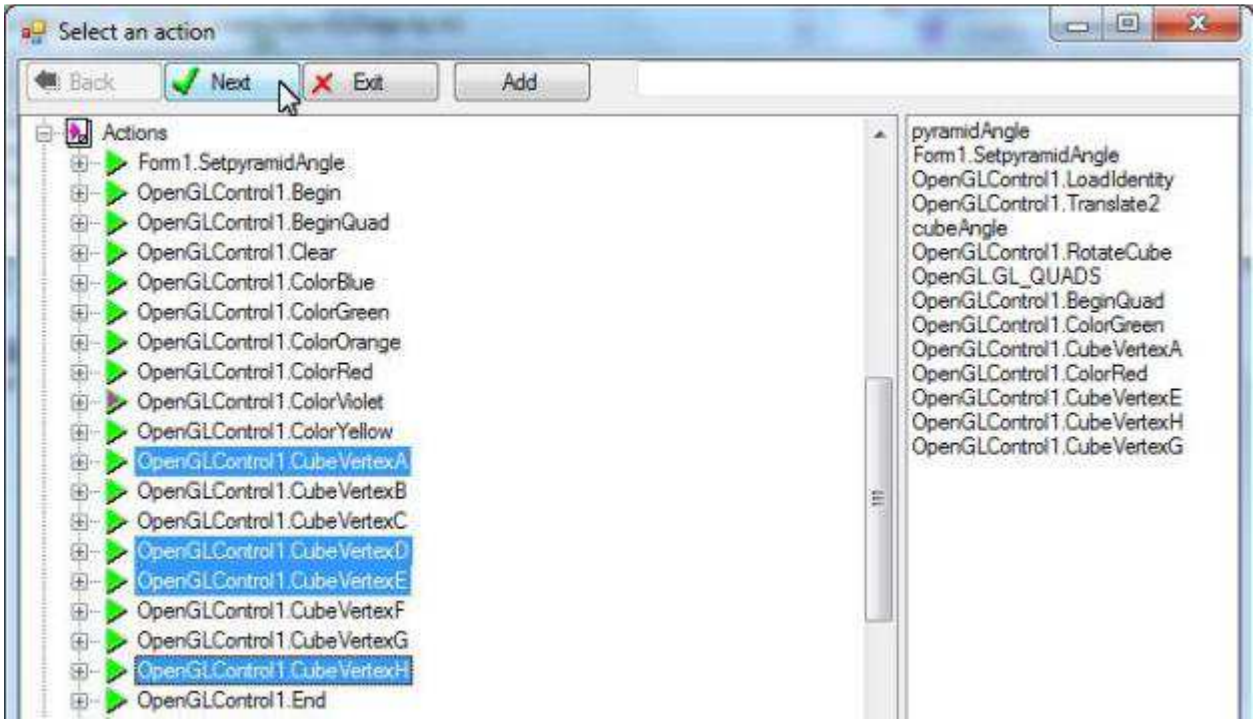


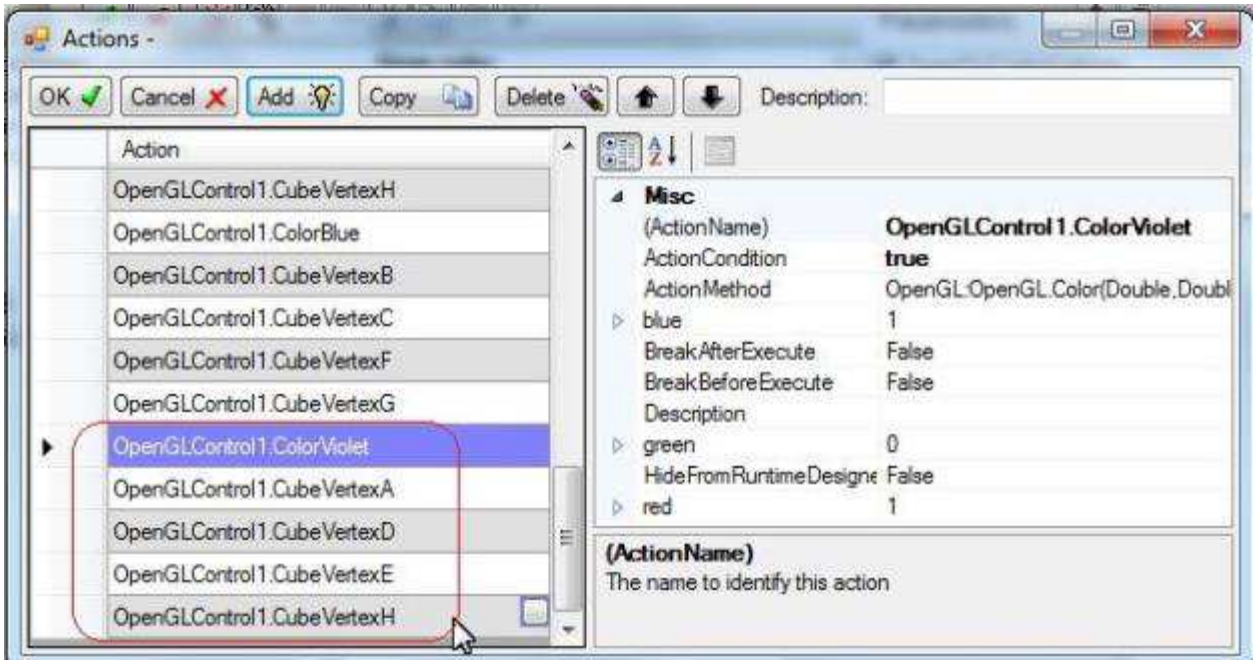
Draw cube side ADEH with color violet

Copy and create a Color action to draw violet (red=1, green=0, blue=1):



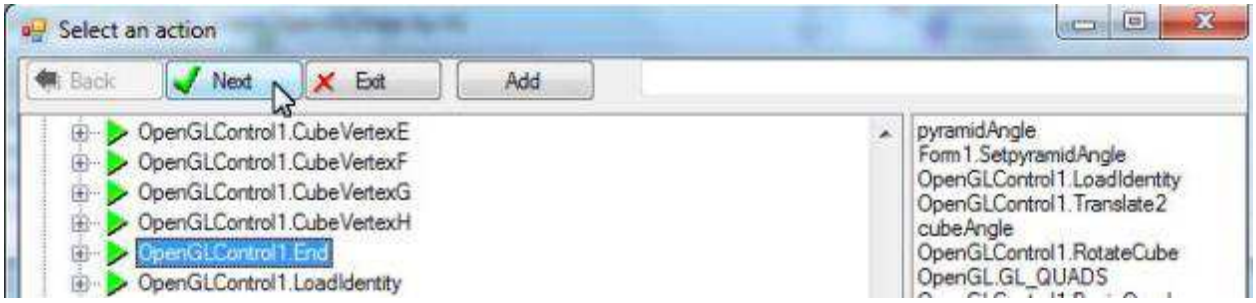
Vertex actions for A, D, E, H are available. Add them to the list:



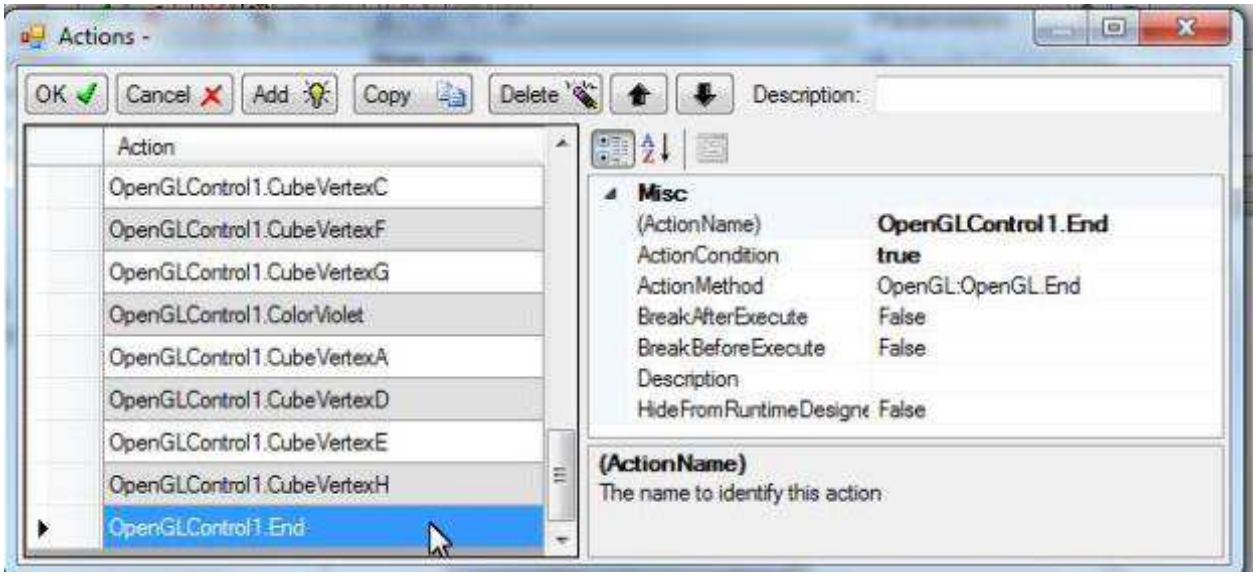


End of drawing quadrilaterals

Add an End action to finish drawing quadrilaterals. We already create the action for finish drawing triangles for the pyramid sides. We may add that action to this list:



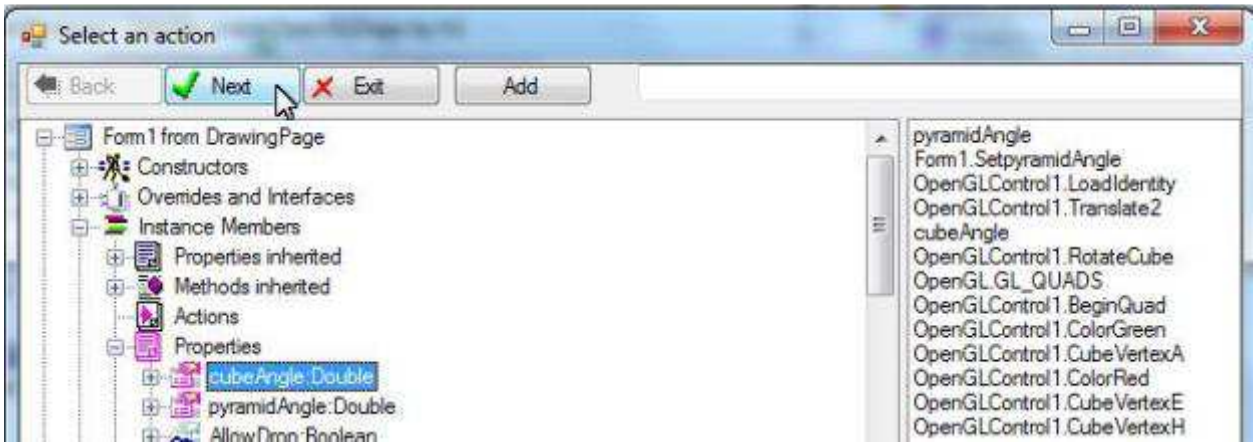
The action appears in the list:



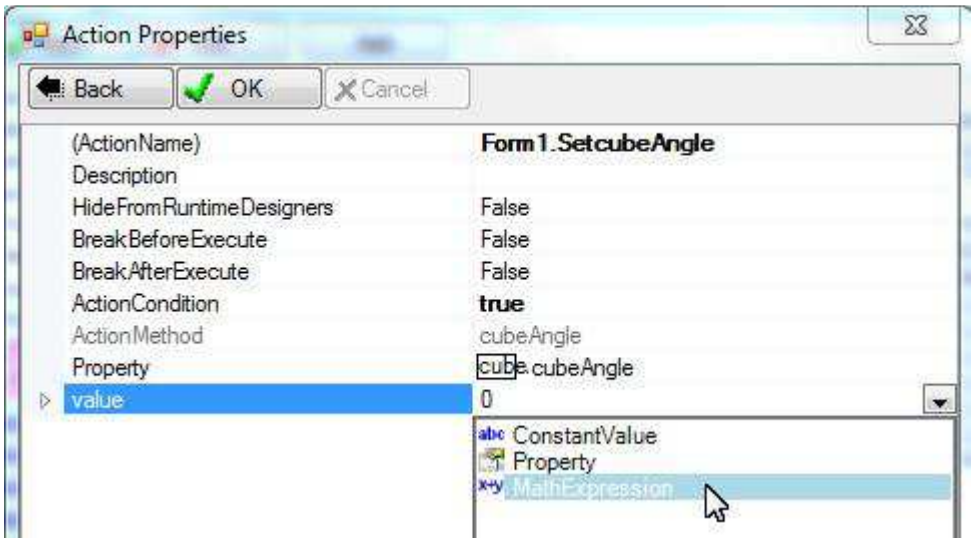
Increase cube rotation angle

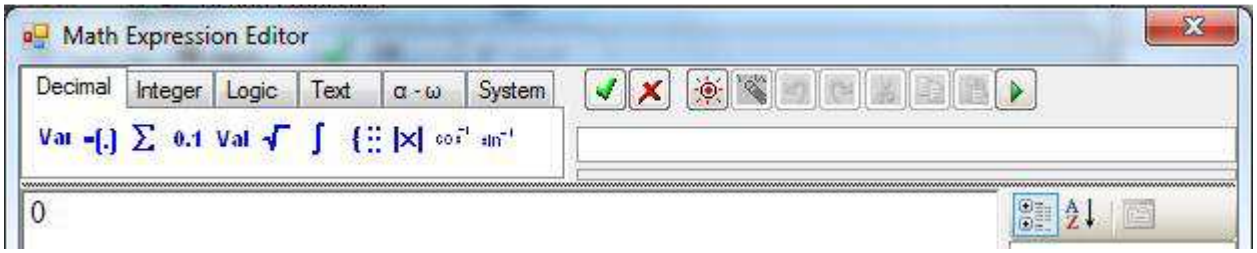
We increase the property, cubeAngle, for the next frame. Click Add:

Select cubeAngle property to change the value of it:

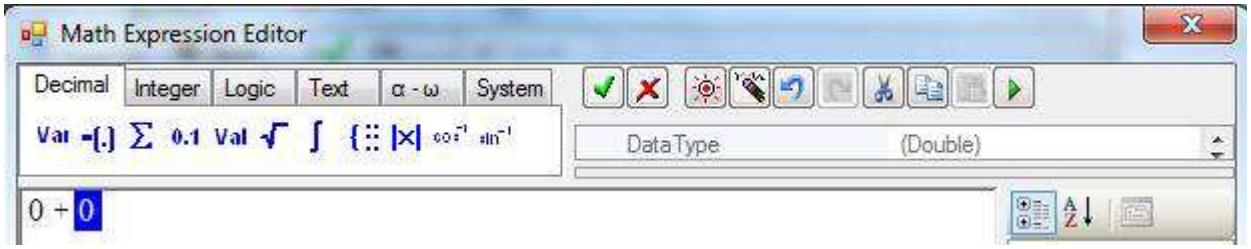


Select Math Expression for the “value” of the action to make calculation for the next cube angle:

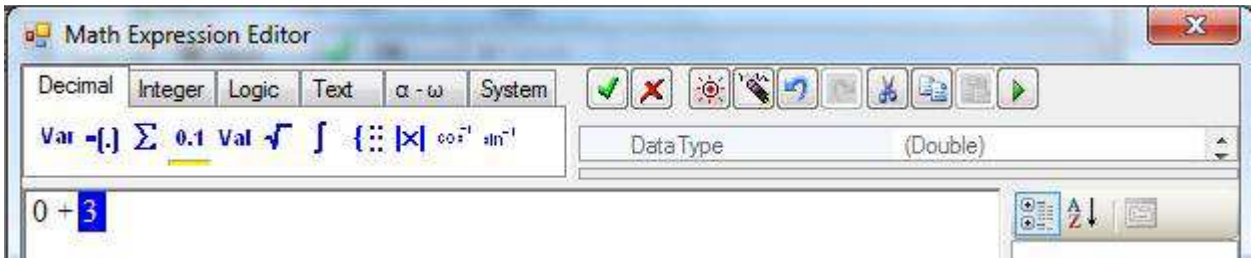




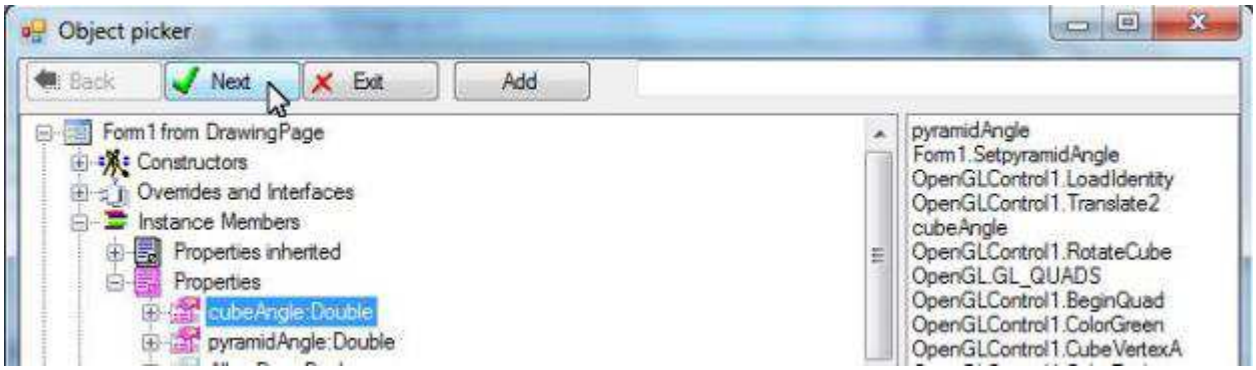
We want to increase the rotation angle by 3 degrees. Press “+” key to create a plus expression:



Press “3” for 3 degrees:

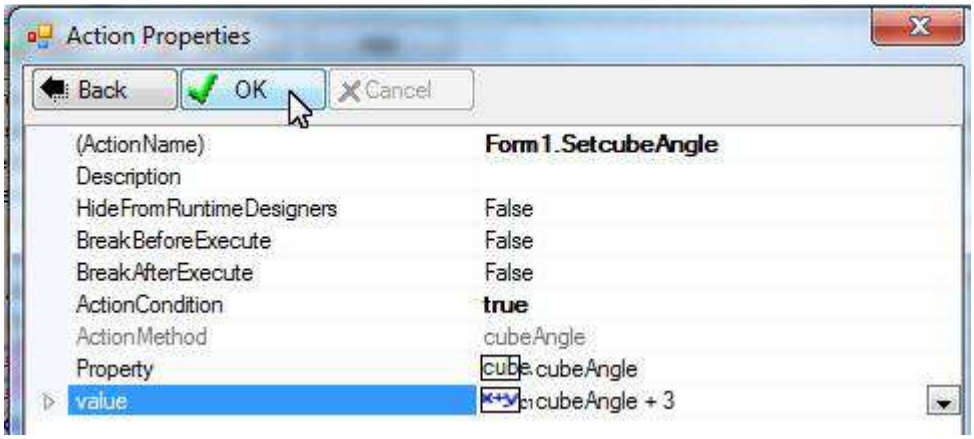


Select the first item and click the Property icon to select the cubeAangle property:

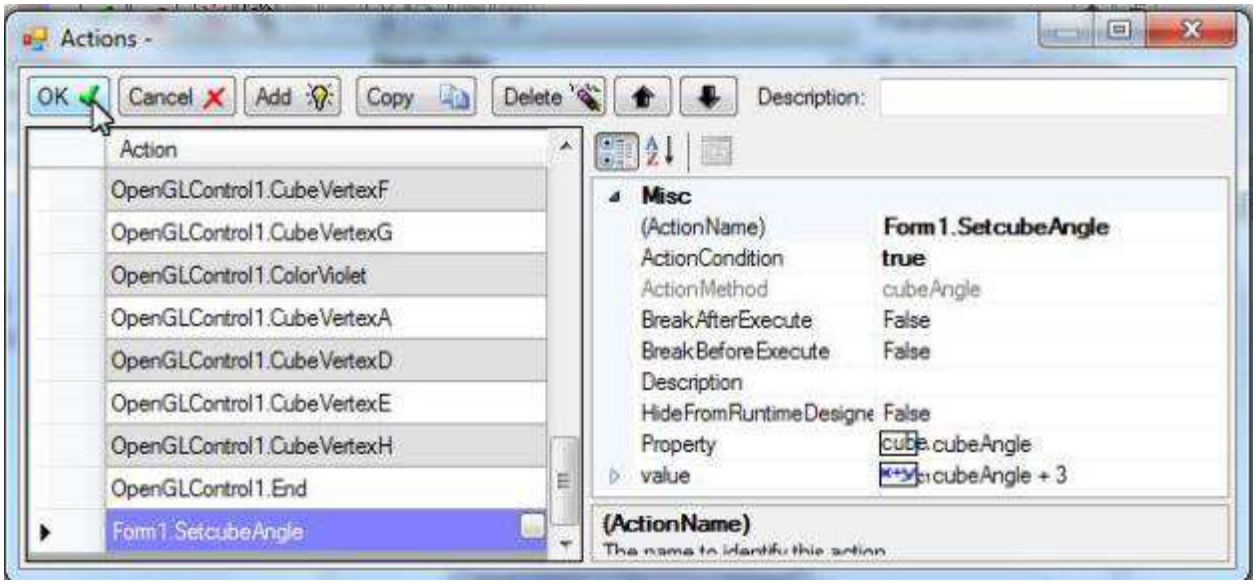


This is the calculation of the next cube rotation angle:

Click OK to finish creating this action:

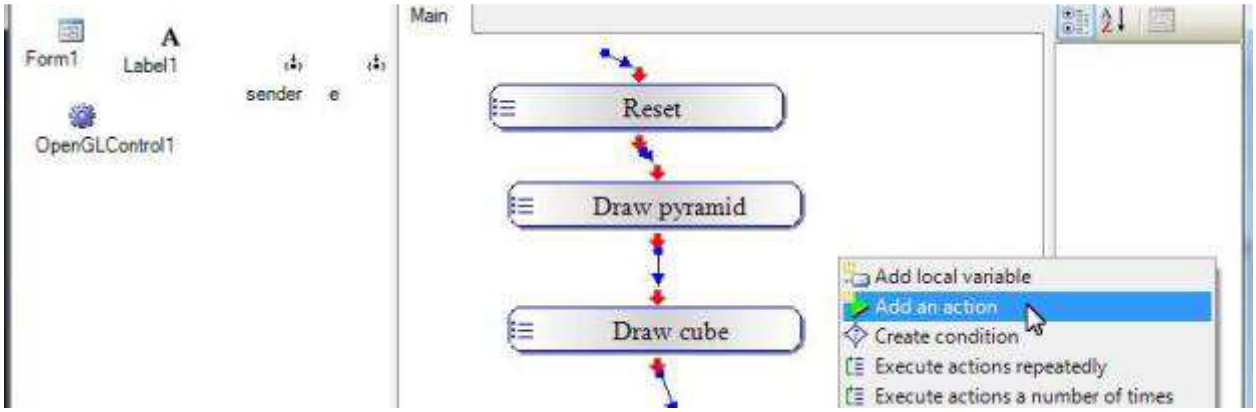


The action appears in the list. The list now contains all the actions needed for drawing the cube:

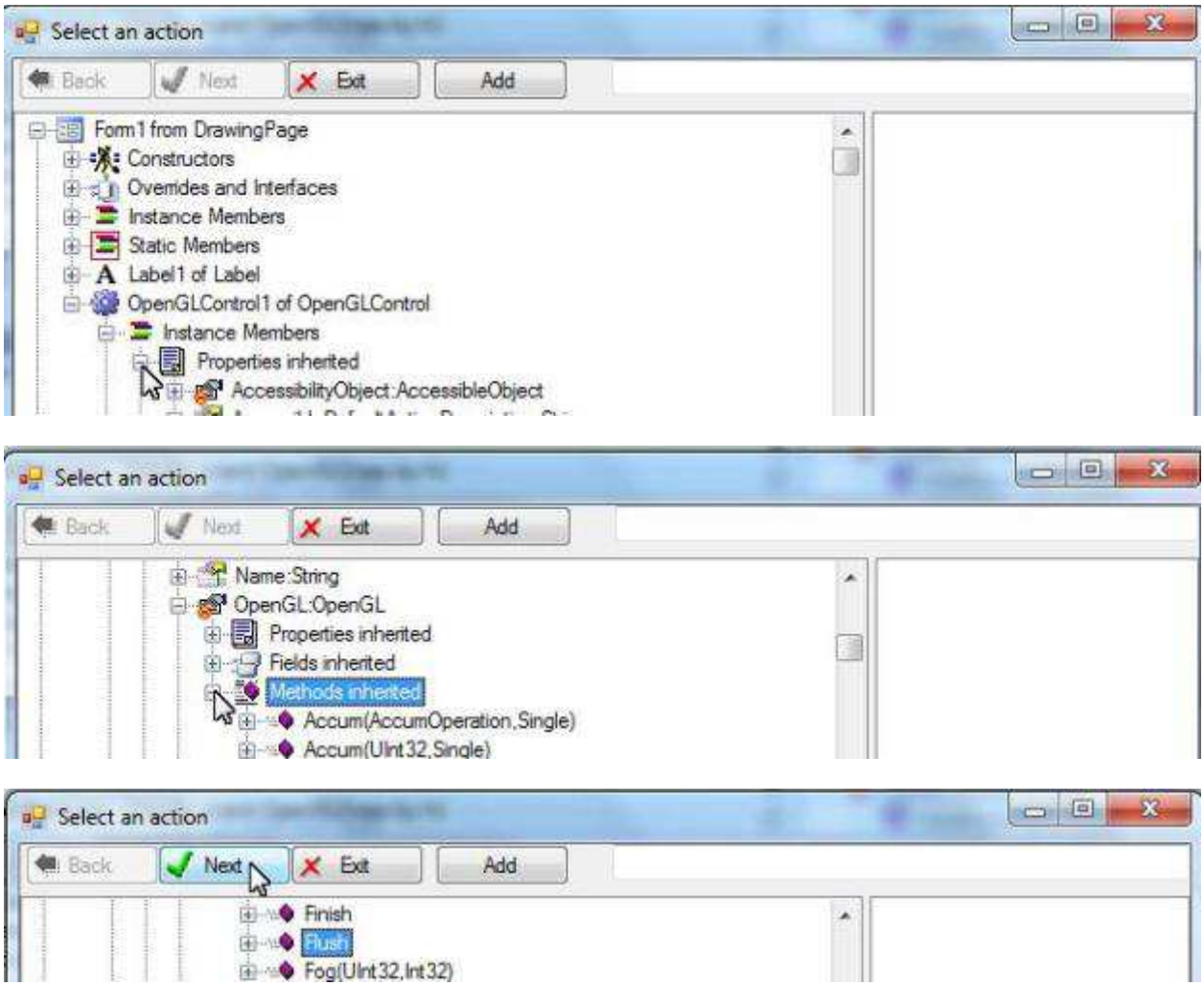


End of handling OpenGLDraw event

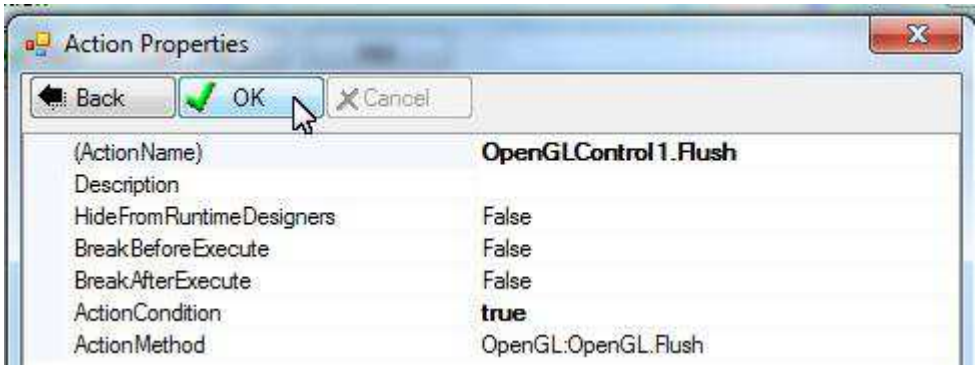
Use a Flush action to force execution of OpenGL functions in finite time:



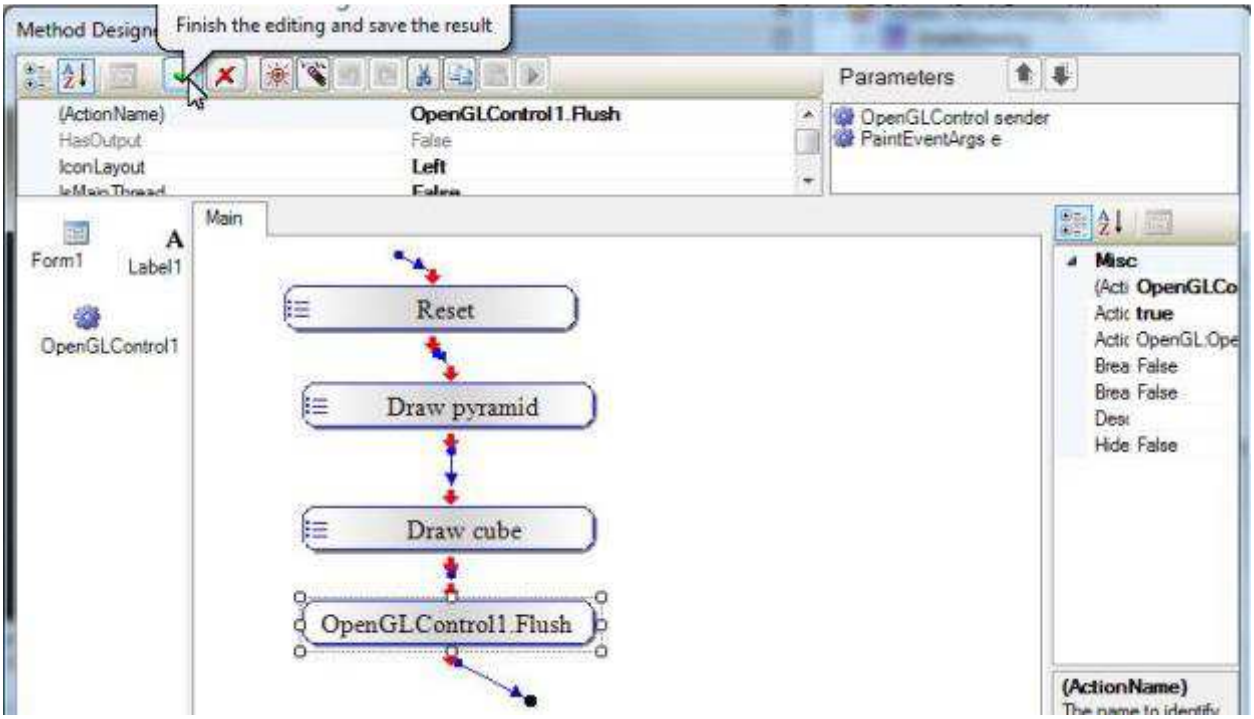
Select Flush method:



Click OK:

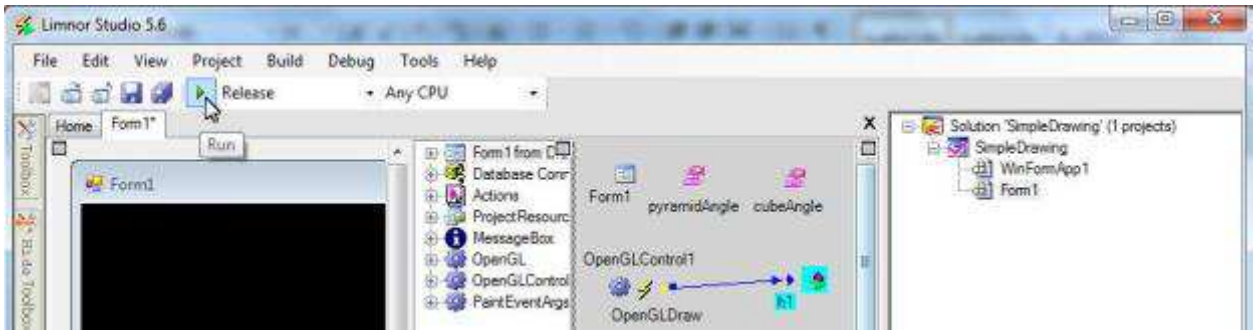


Link it to the last action. We are done creating this event handler method:

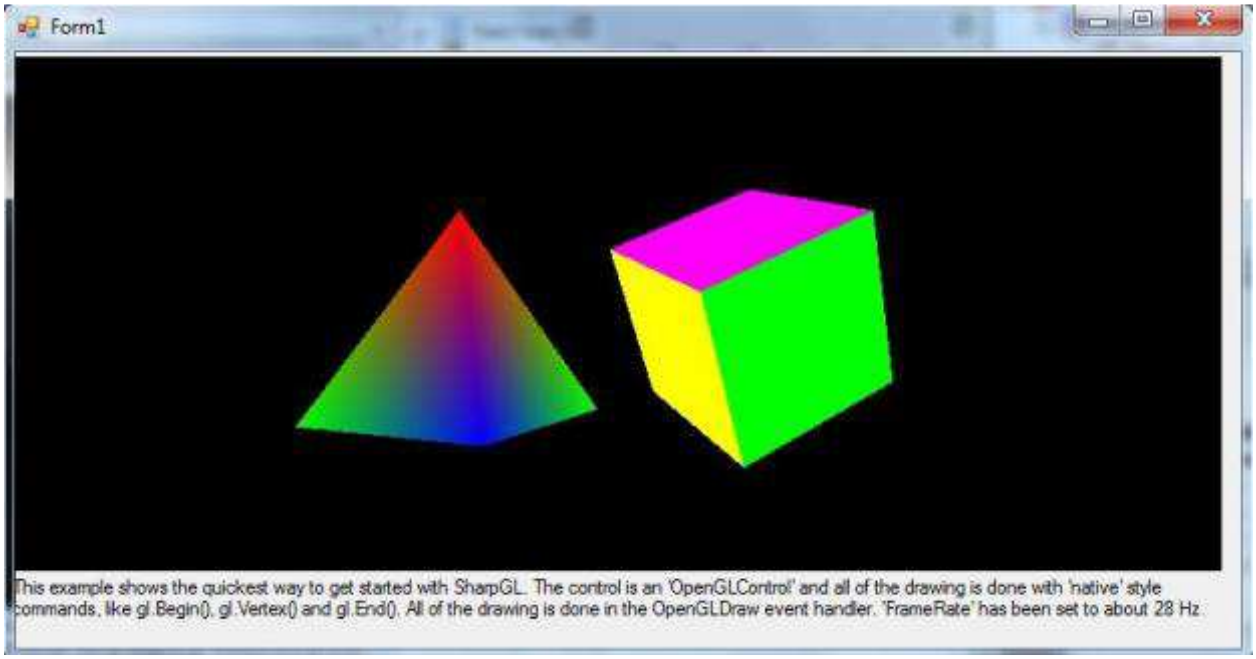


Test

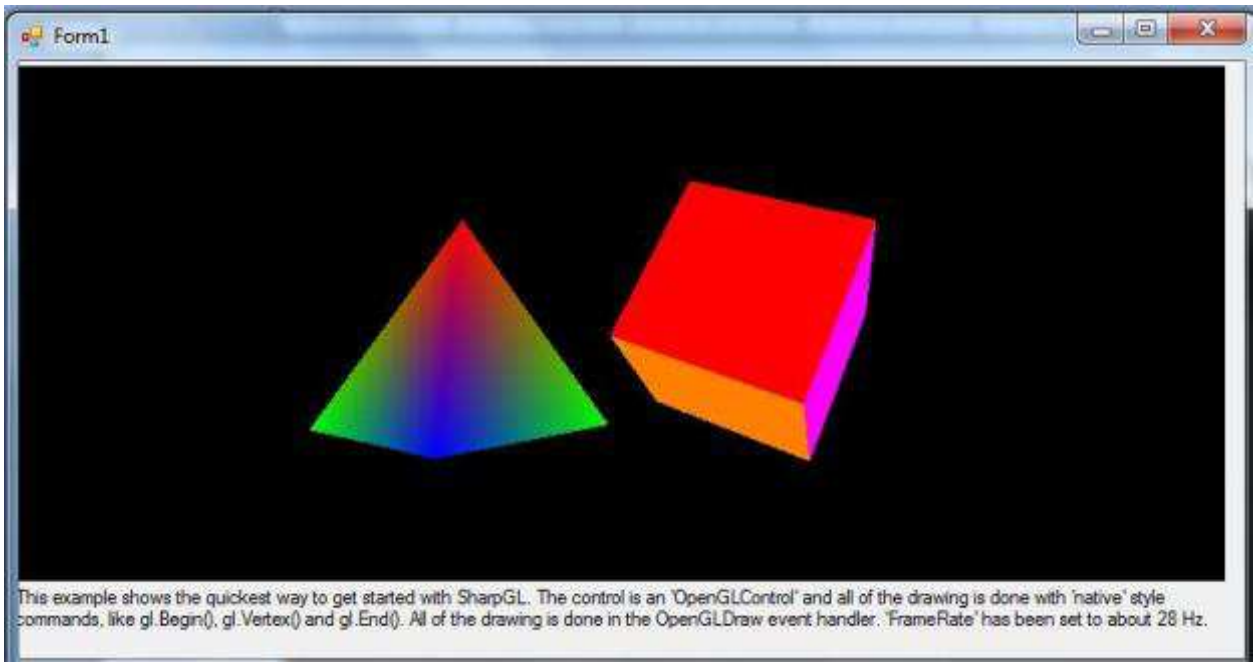
Click the Run button to compile and run the application to see the result:

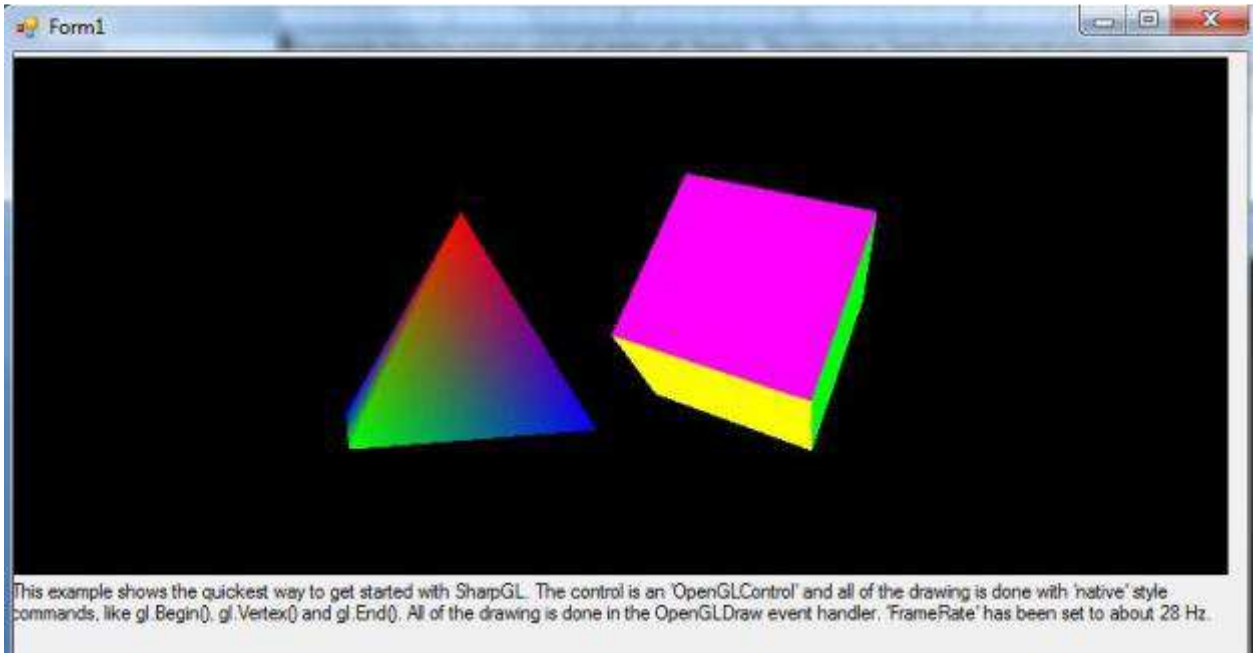


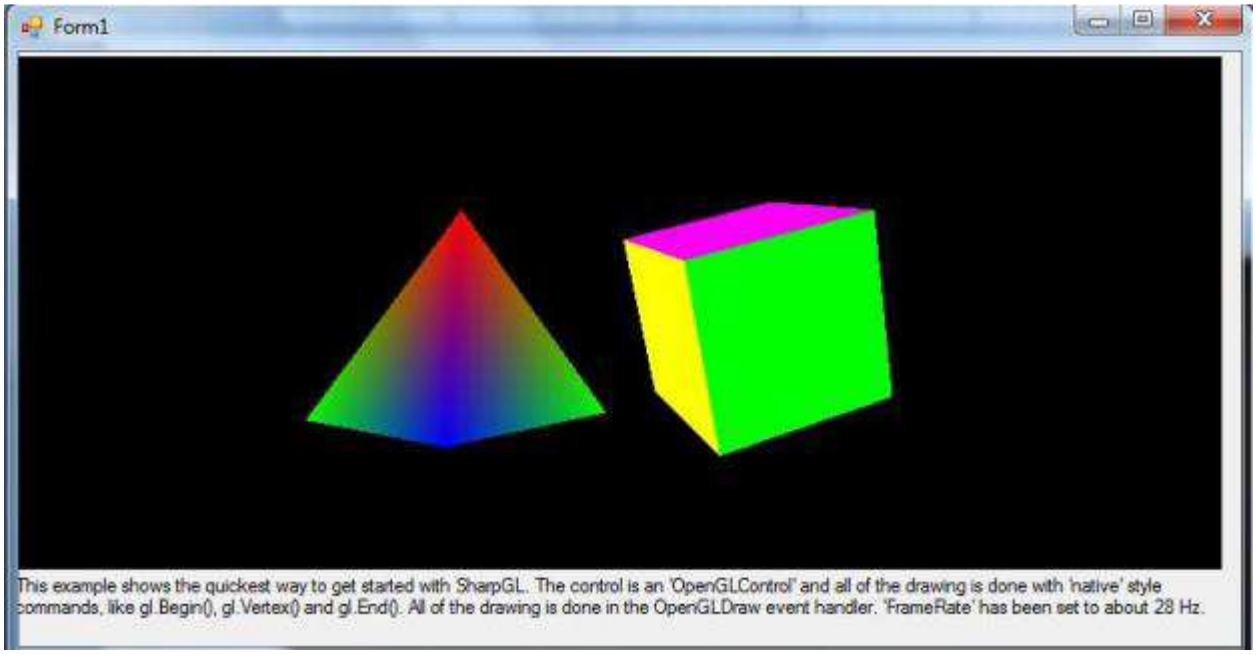
The form appears. A pyramid appears at the left part of the form and a cube appears at the right part of the form:



The pyramid and the cube keep rotating:







The files for this sample project can be downloaded from:

http://www.limnor.com/studio/SharpGL_SimpleDrawing.zip