Study Guide

GETTING STARTED WITH WINDOWS AND MAC DEVELOPMENT

Creating a Windows, Mac and iOS 3D Application

E-Learning Series Course Book

Lesson 7

Embarcadero Technologies

Americas Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

Lesson 7 – Creating a Windows, Mac and iOS 3D Application

Version: 0.8
Presented: June 14, 2012
Last Updated: June 19, 2012
Prepared by: David Intersimone "David I", Embarcadero Technologies

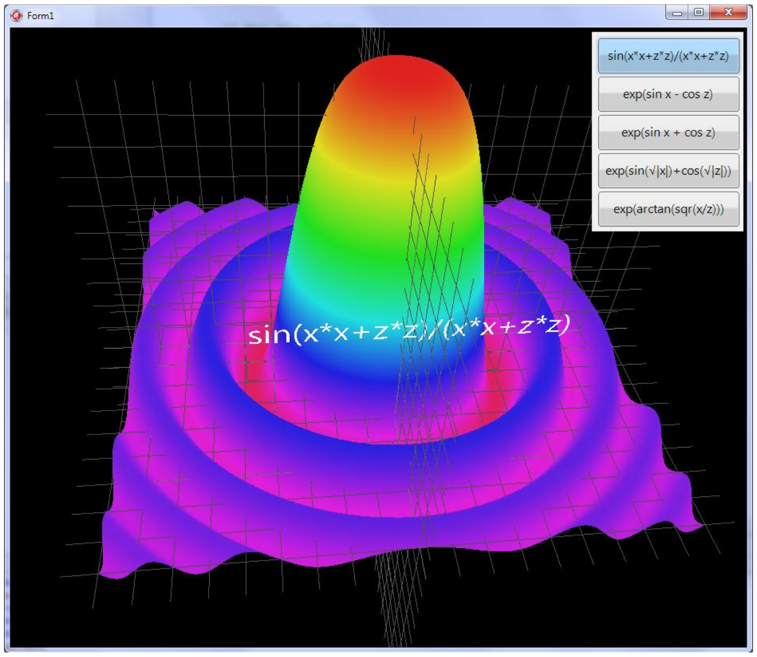
davidi@embarcadero.com
http://blogs.embarcadero.com/davidi/

Contents

Introduction

FireMonkey includes a full suite of drag-and-drop user interface controls. Using FireMonkey you can create 3D applications for Windows, Mac and iOS and add 3D functionality to your HD applications.

FireMonkey 3D Forms provide a full GPU-powered 3D surface with lighting, textures, and animation for highly interactive user interfaces. 3D controls, objects and meshes can be imported from popular 3D design packages with support for file formats such as COLLADA, OBJ and more. A TViewport3D displays 3D content in an otherwise 2D form, while a TForm3D starts with 3D content. 2D and 3D can be nested.



Using FireMonkey 3D requires that your computer has a Graphics Processing Unit (GPU). Most modern computers come with graphics chips that support 3D. On Windows, if the graphics chip has a Direct3D driver, then FireMonkey 3D applications will work. All modern Macintosh computers contain a compatible GPU. All iPhone and iPad devices have GPUs.

In lesson 7 we will focus on designing Windows and Macintosh 3D applications. You can also use all of the capabilities you learned in "Lesson 5 – Designing a FireMonkey HD User Interface" and "Lesson 6 – Connecting to Data" including the use of styles, UI controls and databases. You will find those lesson course books and videos on the "Getting Started with Windows and Mac Development" landing page at http://www.embarcadero.com/firemonkey/firemonkey-e-learning-series.

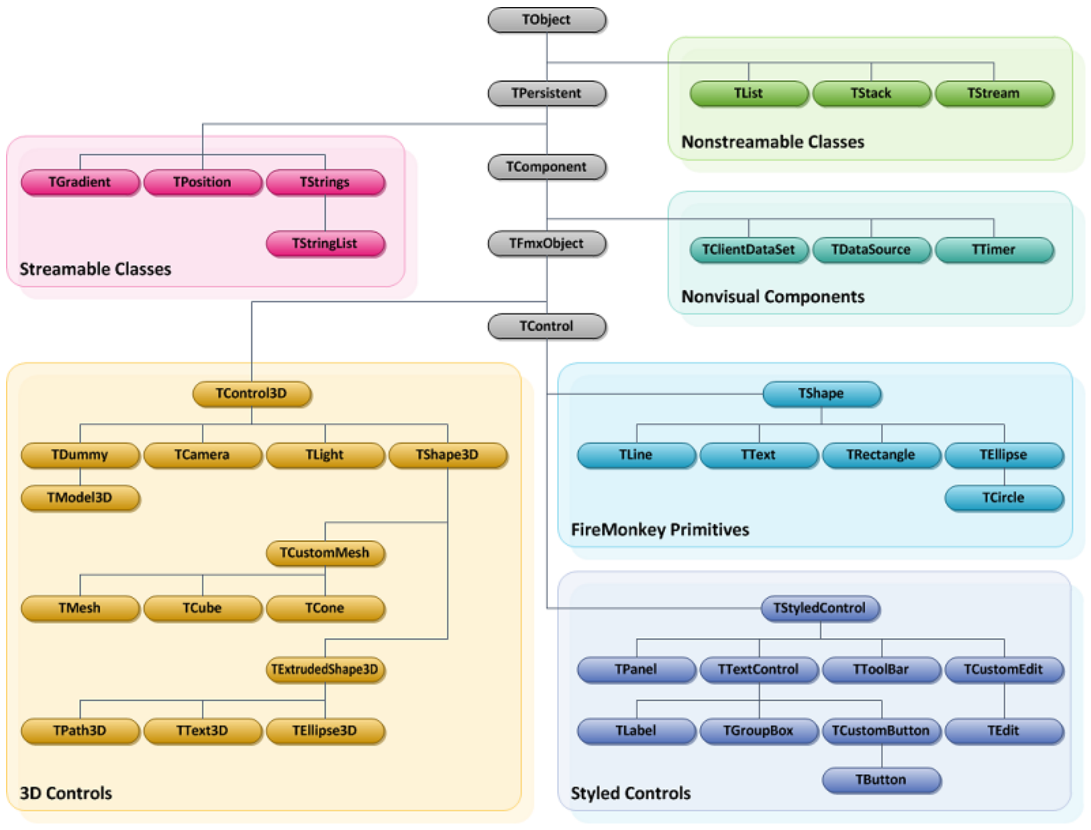
FireMonkey Business Application Platform (FMX)

FMX is the unit scope that contains the units and unit scopes of the Fire Monkey application platform. FireMonkey leverages the graphics processing unit (GPU) in modern desktop and mobile devices to create visually engaging applications on multiple platforms, targeting the entire range from the personal to the enterprise. Major features of Fire Monkey include:

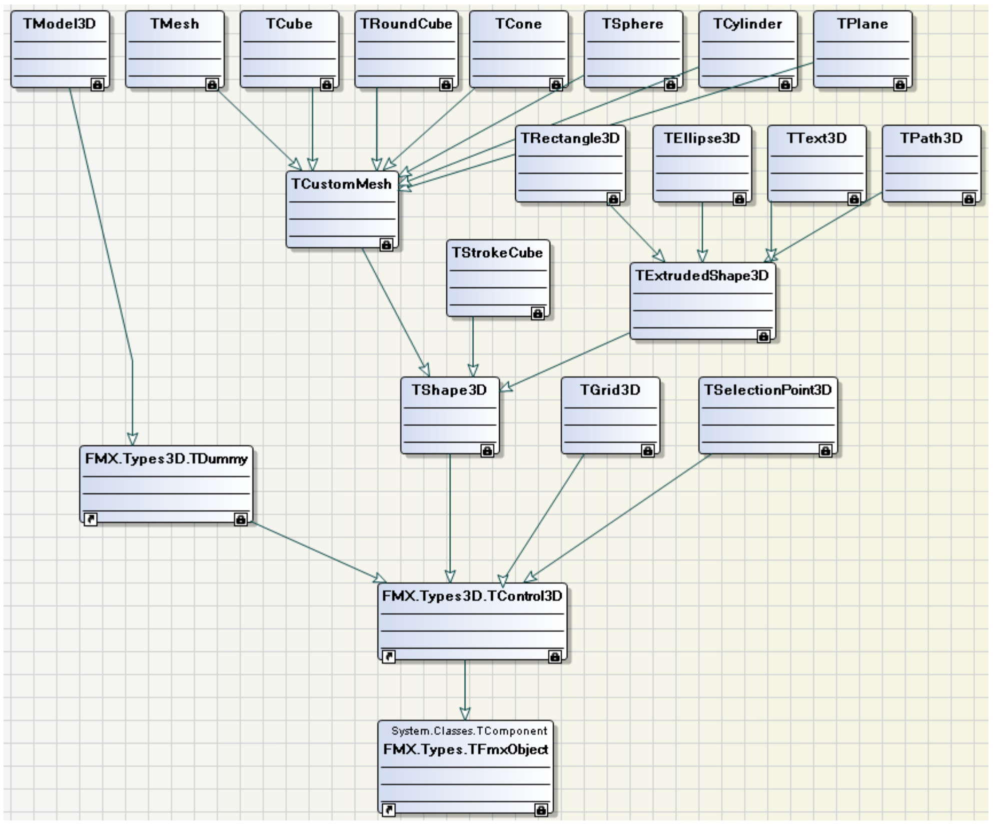
- Cross-platform abstraction layer for OS features like windows, menus, timers, and dialogs
- 2D and 3D graphics
- Powerful vector engine (like Adobe Flash or Microsoft WPF)
- Fast real-time anti-aliased vector graphics; resolution independent, with alpha blending and gradients
- WYSIWYG designer and property editors
- Advanced GUI engine - window, button, textbox, numberbox, memo, anglebox, list box, and more
- Advanced skinning engine based on vector graphics styles with sample style themes

- Shape primitives for 2D graphics along with a built-in set of brushes, pens, geometries, and transforms
- Advanced animations calculated in background thread; easy to use and accurate, with minimal CPU usage and automatic frame rate correction
- Bitmap effects rendered in software, including drop shadows and blurring
- Flexible layouts and compositing of shapes and other controls
- Layered forms, Unicode-enabled
- JPEG, PNG, TIFF, and GIF format read/write support
- Multi-language engine, editor and examples

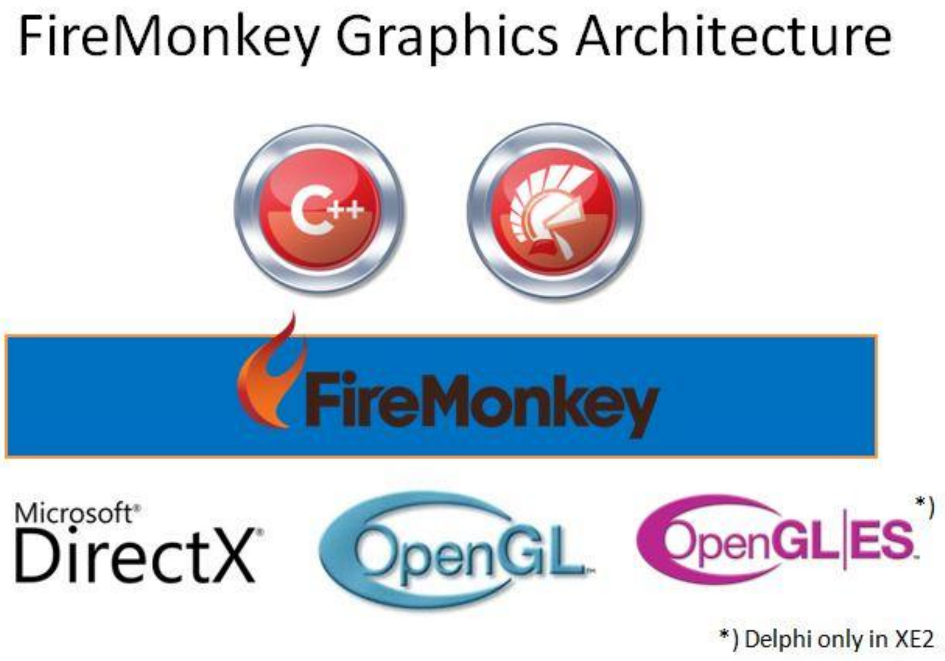
The following figure shows the relationship of some key classes that make up the FireMonkey hierarchy. You can also download a FireMonkey architecture schematic poster (PDF file) at http://www.embarcadero-info.com/firemonkey/firemonkey_chart_poster.pdf.



Here is a UML class diagram for part of FireMonkey's 3D types:



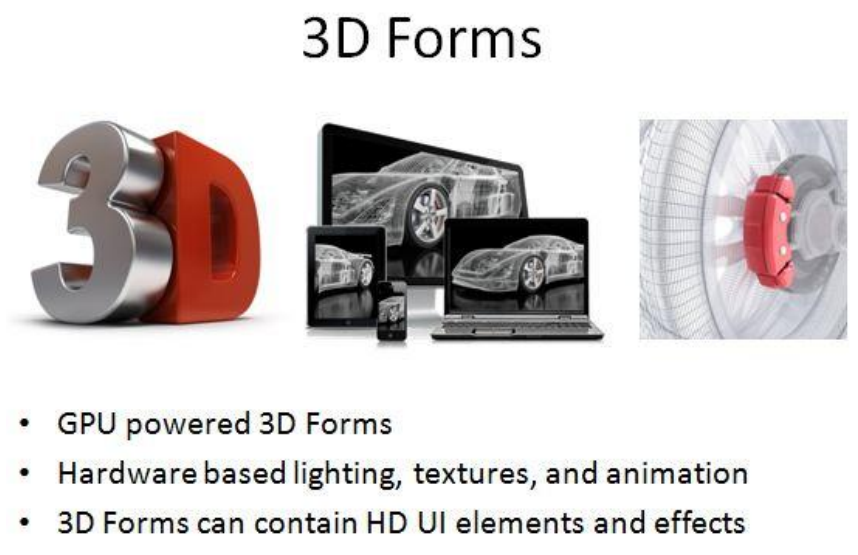
FireMonkey 3D is rendered on Windows using Direct3D, on the Mac using OpenGL and on iOS using OpenGL|ES (Delphi only in XE2).



Using FireMonkey 3D you don't have to worry about differences in the underlying graphics rendering systems on Windows, Mac and iOS. You write your programs using FireMonkey's components and you will get high speed 3D graphics on each platform.

FireMonkey 3D Primer

FireMonkey presents 2D views of objects in 3D space. A TViewport3D displays 3D content in an otherwise 2D form, while a TForm3D starts with 3D content.



3D Forms

- GPU powered 3D Forms
- Hardware based lighting, textures, and animation
- 3D Forms can contain HD UI elements and effects

2D and 3D can be nested. FireMonkey 3D uses platform-specific libraries, as follows:

- For Windows, the Direct3D library (part of DirectX; provided in the Winapi unit scope)
- For Mac, the OpenGL library (provided in the Macapi unit scope)

FireMonkey 3D Coordinate System

Most 3D graphics sub-systems use one of two Cartesian coordinate systems (although there are graphics applications and systems that also use other coordinate systems):

- The (so called) Right Handed coordinate system – X positive is to the right, Y positive is up, Z positive comes out of the screen. OpenGL provides multiple coordinate systems for 3D graphics including Object Space, World (model) Space, Camera Space and Screen Space. For the World Space OpenGL uses the right handed coordinate system.
- The (so called) Left Handed coordinate system – X positive is to the right, Y positive is up, Z positive is into the screen. Microsoft Direct3D (http://msdn.microsoft.com/en-us/library/windows/desktop/bb204853(v=vs.85).aspx) uses the left handed coordinate system.

Since FireMonkey needs to support many types of 3D coordinate systems, it uses its own coordinate system and transforms the X, Y and Z coordinates under the covers so you can write one set of source code for your programs.

The **Position** property of a 3D object or a control contains the X, Y and Z values for the object's position. For a 3D object, the position is relative to 3D space. For a 3D control, the position is relative to its parent. The values for X, Y and Z are single precision floating point types.

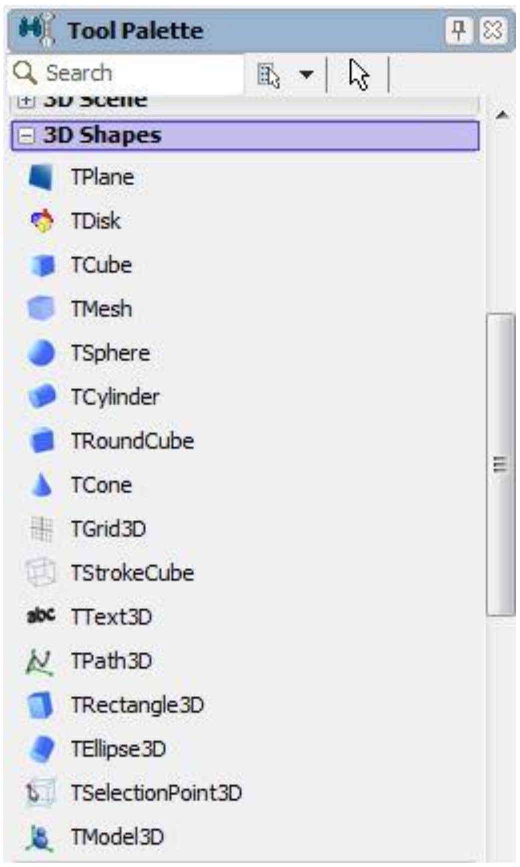
- Positive X – the 3D object to the right of the center of the parent
- Positive Y – the 3D object is below the center of the parent
- Positive Z – the 3D object is away from the user (into the screen)

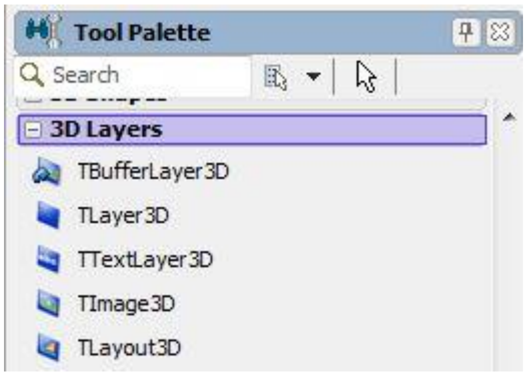
In 3D, rotation is always from the center, with the RotationAngle a TPosition3D, specifying degrees on the X, Y, and Z axes. Rotation also follows the right-hand rule; for example, with X and Y rotation zero, the Z axis points into the screen, and positive rotation on the Z axis rotates clockwise. In 2D, scaling occurs before rotation, which matters because scaling is from the origin and the rotation is adjustable. In 3D, both occur from the center, so the order does not matter.

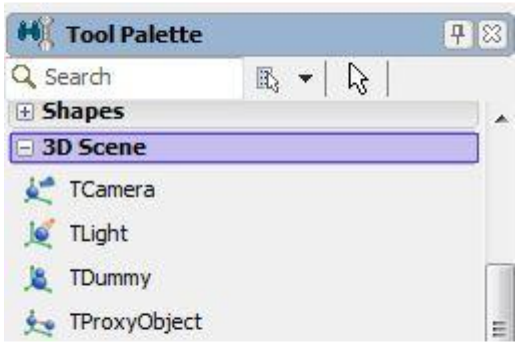
3D Objects

FireMonkey provides several kinds of 3D objects:

- Primitive 3D shapes like TCube, TSphere, TCylinder and TCone.
- 2D objects extruded into 3D, like TText3D and TPath3D. Extruded objects have three sides: the front, with the original 2D shape; the back, a mirror image of that shape; and the shaft of the extrusion between them.
- Flat 2D objects in 3D space, like TImage3D and TTextLayer3D.
- TMesh for complex 3D objects.
- TModel3D to load 3D model files for .OBJ, .DAE and .ASE model formats.







Objects are positioned in 3D space with X, Y, and Z coordinates using a TPosition3D. In addition to Height and Width, they also have a Depth. Flat 2D objects in 3D space are arbitrarily thin with a hard-coded Depth of 0.01. 3D objects have 3D rotation and scaling. Position, size, rotation, and scaling are all relative to the object's center point.

As with 2D controls, any 3D control in FireMonkey can be the parent of any other 3D control. A child's position and rotation is relative to its parent. Moving or rotating a parent will re-position its component sub-tree. TLayout3D may be used as an otherwise featureless parent to organize other objects.

While 2D objects arranged on a HD surface require a notion of Z-order to determine their layering, objects in 3D space are inherently ordered, so that when seen from a given vantage point, nearer objects will occlude farther ones in the same line of sight.

Cameras

Every view of 3D space is controlled by a camera. The position and orientation (3D rotation) of the camera determines what you see. There is always the **design camera** used in the Form Designer, and by default at runtime. The design camera is directly above the negative-Z axis (toward negative-Y), perpendicular to the X-Z plane, angled slightly downward so that position 0,0,0 is in the center of the view. To use a different camera, set the **UsingDesignCamera** property of the TViewport3D or TForm3D to False, and assign a TCamera to the Camera property. A scene may have multiple cameras. After assigning a different one to the Camera property, you must call Repaint manually.

The viewable volume is a frustum, with the near plane slightly in front of the camera. Anything between the camera and that front plane is clipped from view. The angle of the field of view is fixed vertically: making the viewport/form taller makes everything bigger, while making it wider shows more to the sides. The units of position and size are relative to the scale of the field of view.

Screen Projection

In addition to an object's view (or absence from view) being determined by the camera, an object can also be set to be seen no matter where the camera points, like fixed status indicators in 3D games. This is done by changing the Projection property from the default pjCamera to pjScreen, so that:

- At Position.Z zero:
    - XY 0,0 is the top left corner, just like with 2D. (3D position still reflects the position of the center of the object, not its top-left as with 2D.)
    - The Height, Width, and Depth dimensions are equal to pixels, no matter how the field of view is scaled.
    - Because the units are rendered as pixels, dimensions of an object using screen projection are much larger than when using camera projection to appear as the same size.
    - The object appears slightly behind the front clipping plane of the frustum. Objects using camera projection have a small gap where they can appear in front of screen projected objects before being clipped by the front plane.
- At Position.Z greater than zero (away from the viewer):
    - Rendered size is smaller.
    - Shrinking the view pushes the object farther back, making it even smaller.
    - Enlarging the view brings the object closer, making it larger.
- At Position.Z less than zero (toward the viewer):
    - Rendered size is larger.
    - Shrinking the view brings the object closer, making it even larger.
    - Bring it too close, either by shrinking the view or setting Position.Z directly, and the object will move past the front plane of the frustum, causing it to disappear.
    - Enlarging the view pushes the object farther back, making it smaller.

In the Form Designer, changing an object's Projection recomputes its position so that it does not move, and also adjusts the size.

Camera Boom

Viewing an object or scene from any direction, controlled by the user, is a common 3D application. Although you can compute the position and orientation to achieve the proper camera angle, it is easier to create a virtual camera boom:

1. Create an invisible object, like a TDummy, in the same location as the object of interest, or the center of the scene.
2. Create the TCamera as a child of the object.
   a. Set its Position the desired distance away on one of the axes.
   b. If necessary, set its RotationAngle so that the camera points back along the axis to the location. Make sure the camera is not upside-down.
3. You can now rotate the camera around the object simply by changing the RotationAngle of the dummy object. The camera maintains the exact distance and automatically points directly toward the center.
4. Adding a light as a child of the camera will maintain the distance and orientation of the light as the camera moves.

Lighting

In general, a 3D object is a featureless black mass unless it has some light on it. One or more TLight objects define light in the 3D space, depending on their LightType property:

- ltDirectional - Directional light is constant, from a given angle. Light from the sun is a common analog: it is from very far away, so for any localized spot on earth -- yards across, not miles -- all items are lit the same. The position of a directional light is irrelevant to its effect. (Its position is relevant if you try to click it in the Form Designer.) What matters is the direction the light points, as defined by its RotationAngle, and the RotationAngle of its parents.
- ltPoint - Point light is like a bare light bulb (with no stem). It radiates in all directions, and falls off with distance. Its RotationAngle has no effect. What matters is its position, which is affected by the position and rotation of its parents.
- ltSpot - Spot light depends on both position and rotation, and falls off with distance.
- Flat 2D objects in 3D space neither require, nor are affected by light. They appear as usual with added 3D perspective.

Materials

The surface of a 3D object is defined by its material. Primitive 3D objects have a single Material property. Extruded objects have additional MaterialBack and MaterialShaft properties. Each is of type TMaterial. Several of its properties are colors that control how the surface appears with light:

- **Emissive** color determines whether a surface emits its own light, or glows. It defaults to null (zero-opacity black): objects normally do not glow, and require light to be seen. By setting a color, surfaces with no light will appear with that color. When lit, the emissive color will blend with the other colors resulting from light.
- **Ambient** color is meant to provide a base color to surfaces so that they may be seen. In the real world, light is reflected from many directions onto a surface; but in a 3D scene it would be difficult to define all that light. So Ambient color is activated by any light in the space. (With no lights, it has no effect.) The surface will be colored fairly uniformly in that color. With any directional light -- which does not have to be pointing at the surface -- everything will look flat. With point or spot lights, the color attenuates with distance.
- **Diffuse** color interacts directly with light, including the angle of incidence. With no light, it has no effect. It is common to set Ambient and Diffuse to the same color.
- **Specular** color simulates a glossy surface, by reflecting incident light at a specific angle, instead of diffusing light in many angles. With no light, it has no effect. It defaults to white to reflect the light without altering its color.

Subdivisions – Width, Height, Depth

For 3D objects like TSphere, TCode, TCube, TCylinder, etc you can control the smoothnes of the surface by setting the SubdivisionsWidth, SubdivisionsHeight and SubdivisionsDepth properties.

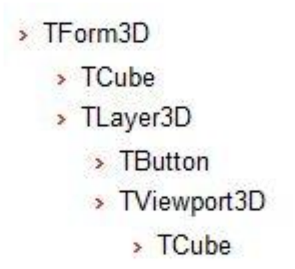
The , SubdivisionsWidth, SubdivisionsHeight and SubdivisionsDepth properties specifies how many identical subdivisions the 3D object will have. A bigger number of subdivisions generates a smoother surface. The default value for each property is 1.

Mixing 2D and 3D

3D objects must be placed in a 3D container. A 3D object (like a TCube) will not render if placed directly in a 2D container (like a TForm). Conversely, 2D objects (like a TButton) will not render if placed directly in a 3D container (like a TForm3D). Two classes allow nested mixing of 2D and 3D content:

- TLayer3D is a 3D object that contains 2D content. It is like a rectangular sheet of glass that lives in 3D space.
- TViewport3D is a 2D object that contains 3D content. Like TForm3D, it is a "window" into 3D space.

You may nest these containers. For example, you can have an object hierarchy like:



In addition, a few classes directly host 2D content in 3D:

- TTextLayer3D displays text
- TImage3D displays bitmap images

TDummy

Represents a class for 3D dummy objects. TDummy is an ancestor class for TModel3D and is internally used by TViewport3D. TDummy can also be used as a container for 3D objects. Use a TDummy control to group other 3D controls so that they can be moved and rotated together.

TProxyObject

TProxyObject is an abstract container class for proxy objects. TProxyObject contains a SourceObject field which is the object that the TProxyObject represents. You can use TProxyObject to create 3D scenes dynamically in code. Using TProxyObject can save memory and improve execution speed since graphics intensive operations only have to be done on the source object.

Font Support

In FireMonkey, Font sizes are expressed in Device Independent Pixels (DIPs). With OS X, the point value is equivalent to pixels, because the display defaults to 72 dots (or pixels) per inch, or DPI. Windows text (at the DirectWrite API level) is sized with device-independent pixels or DIPs, at 96 per inch; and the display also defaults to 96 DPI. So, in both cases, there is a one-to-one correspondence between the font size units and display resolution. Text will be the same height on both Windows and OS X. (The way each platform renders fonts may cause subtle differences.)

For example, suppose the font's Size is 24. On Windows, 24 DIPs is 24/96 or 1/4 inches tall. 1/4-inch on a screen at 96 DPI is 24 pixels. On OS X, 24 points is 24/72 or 1/3 inches tall. 1/3-inch on a screen at 72 DPI is 24 pixels.

Text sized in points on Windows will appear larger at the same numeric value. For example, 24 points at 96 DPI is 32 pixels tall.

You set the Font's **Family** property to specify the typeface of the font. Use the font's **Style** property to add special characteristics to characters that use the font. Style is a set containing zero or more values from the following:

- fsBold - The font is bold.
- fsItalic - The font is italic.
- fsUnderline - The font is underlined.
- fsStrikeOut - The font is displayed with a horizontal line through it.

Creating Your First Windows and Mac 3D Application

Use the following steps to build your first Windows and Mac 3D application.

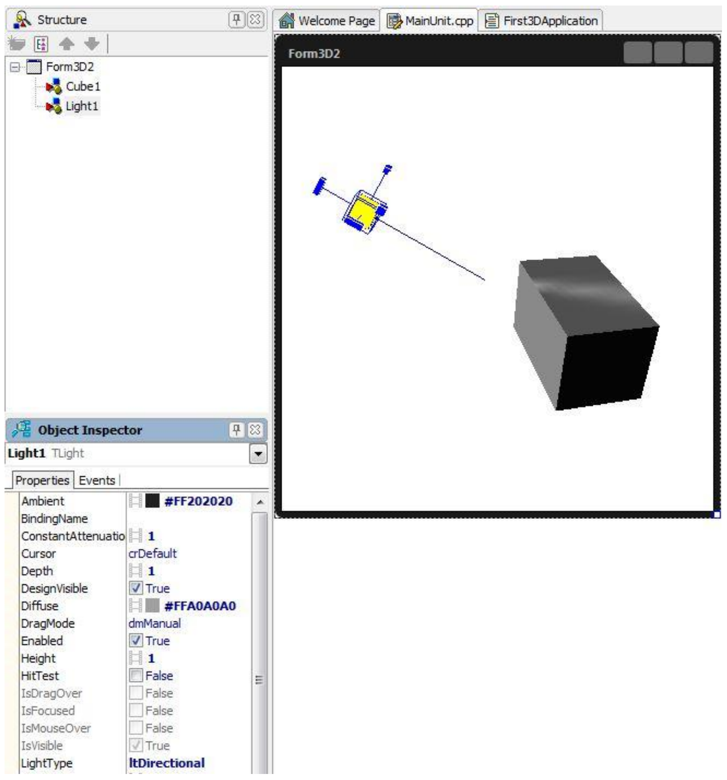
Step 1: Using **File > New > Other > C++Builder Projects > FireMonkey 3D Application** and **File > New > Other > Delphi Projects > FireMonkey 3D Application** creates the starting project for a FireMonkey 3D application and opens the FireMonkey Form Designer, displaying the base form (FMX.Forms.TForm3D).

TForm3D represents a standard FireMonkey 3D application window. 3D forms can represent the application's main window, dialog boxes, or other. A 3D form can contain FireMonkey 3D objects, such as TText3D, TGrid3D, and so on. All the objects that can be employed in 3D forms reside in the Objects3D unit.

Step 2: From the Tool Palette, add the following two FireMonkey 3D components (by using IDE Insight or entering the component name in the Search field and pressing Return):

- TLight
- TCube

Separate the two components (by dragging the TLight component into the upper left corner of the form). You can set TLight's LightType property to ltDirectional, ltPoint and ltSpot. I set LightType to the default value ltDirectional. Here is the IDE with the 3D form and its two child components:



Ambient	■ #FF202020
BindingName	
ConstantAttenuatio	1
Cursor	crDefault
Depth	1
DesignVisible	✓ True
Diffuse	■ #FFA0A0A0
DragMode	dmManual
Enabled	✓ True
Height	1
HitTest	☐ False
IsDragOver	☐ False
IsFocused	☐ False
IsMouseOver	☐ False
IsVisible	✓ True
LightType	ltDirectional

Step 3: In the FireMonkey Form Designer, adjust the position and size of the FireMonkey 3D form and components to suit your needs:

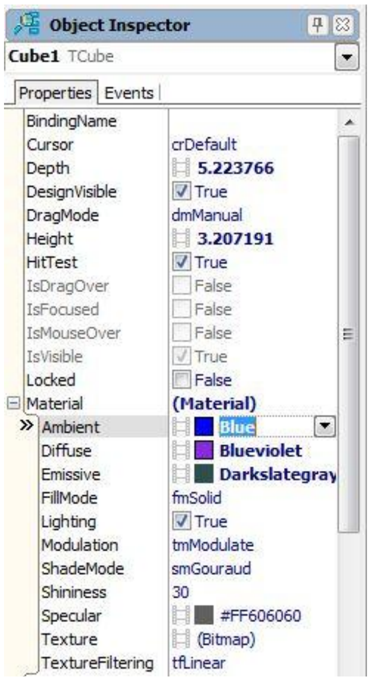
- To move an object, simply drag and drop.
- To rotate a 3D component, use the three blue handles that appear when you click the component. Each handle rotates the component in the associated plane in space (that is, the x, y, or z vertex). When you click a handle, it becomes red to indicate that it is the active handle. Note: You can also use the RotationAngle properties in the Object Inspector (x, y, and z)
- To resize a component, use the resizer control, which is a small blue cube located on one corner of the component. The resizer works in one dimension at a time; the resizer is red on the si de of the cube that is the selected (active) dimension. Note: You can also use the properties in the Object Inspector (for TCube, the size-related properties are Depth, Width, and Height).

In the picture included at the end of step 2, I've set the RotationAngle of the light to point down at the cube. I've resized the cube to make it larger and rotated it to the left.

Step 4: Using the Object Inspector, adjust the property settings for your TCube component:

- To specify the color and texture of the TCube, use the Material property.
- You can set the Material's Ambient, Diffuse and Emissive sub-properties by clicking the plus sign to the left of the Material property in the Object Inspector. You can also use the graphical Material Designer by clicking the ellipsis [...] in the Material field.

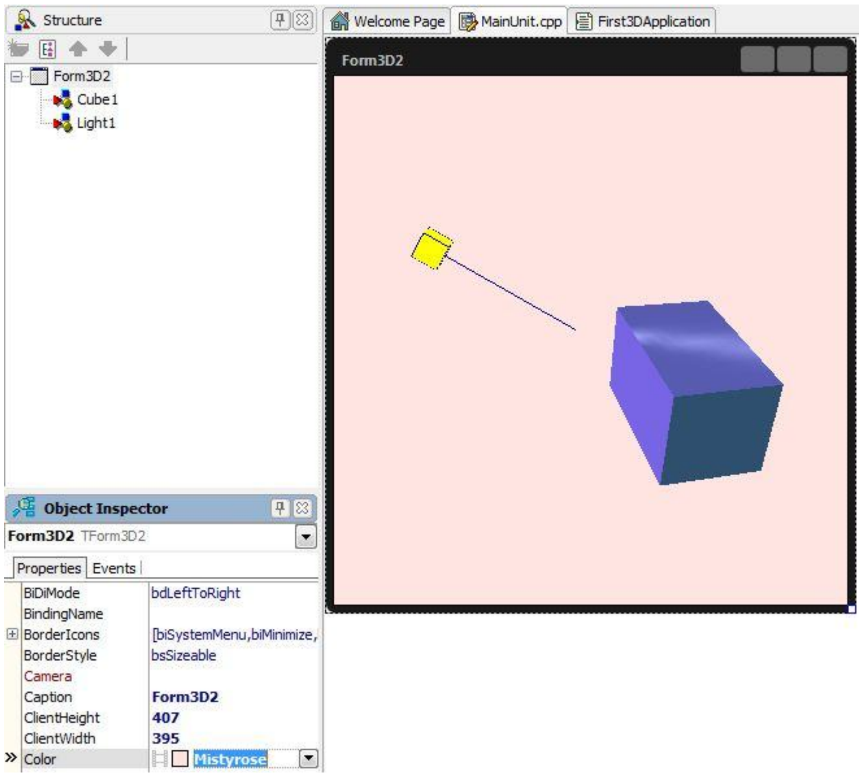
In my example, I've set the Ambient, Diffuse and Emissive sub-property colors of the cube's Material property to Blue, BlueViolet and DarkSlateGray respectively.



Cube1 TCube

BindingName	
Cursor	crDefault
Depth	5.223766
DesignVisible	✓ True
DragMode	dmManual
Height	3.207191
HitTest	✓ True
IsDragOver	False
IsFocused	False
IsMouseOver	False
IsVisible	✓ True
Locked	False
Material	(Material)
Ambient	Blue
Diffuse	Blueviolet
Emissive	Darkslategray
FillMode	fmSolid
Lighting	✓ True
Modulation	tmModulate
ShadeMode	smGouraud
Shininess	30
Specular	#FF606060
Texture	(Bitmap)
TextureFiltering	tfLinear

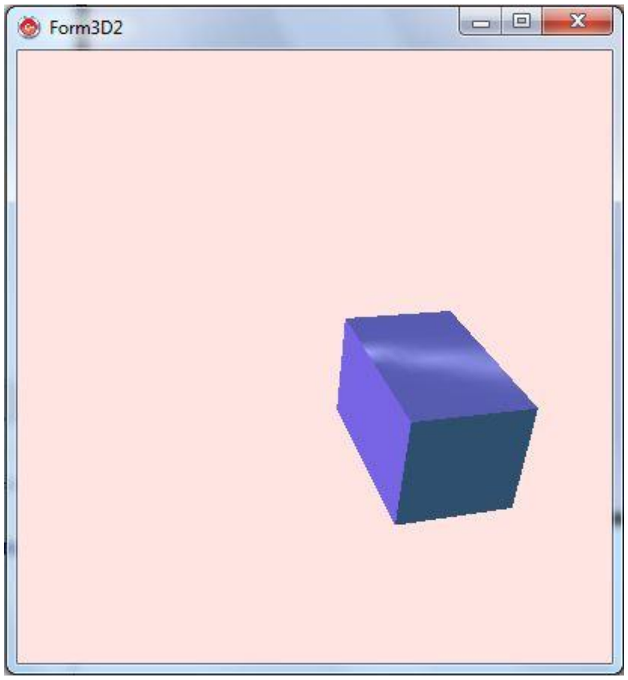
Step 5: In the Object Inspector, change the Color property for the Form. I set my Form Color property to MistyRose. In the following figure, you can see the repositioned light and cube components, two of the three handles and the resizer on the cube, as well as the various IDE panes around the FireMonke y Form Designer:



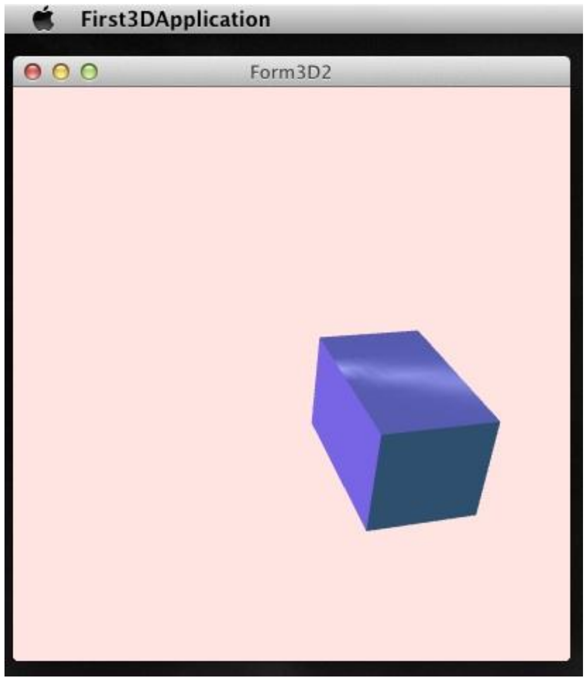
Form3D2 TForm3D2

BiDiMode	bdLeftToRight
BindingName	
BorderIcons	[biSystemMenu,biMinimize,
BorderStyle	bsSizeable
Camera	
Caption	Form3D2
ClientHeight	407
ClientWidth	395
Color	Mistyrose

Step 6: Save your project and hit F9 to run the application in Windows. The running application should look something like the following on Windows:



Step 7: Add OSX to your target platforms in the Project Manager (remember to have your PAServer running on the Mac). Hit F9 to run the application on your Mac. The running application should look something like the following on the Mac:



Congratulations! You've built your first 3D application for Windows and Mac. Remember to save the project. We are going to extend it to add an HD interface to the project in the next section.

Creating a HD Interface in a 3D Application

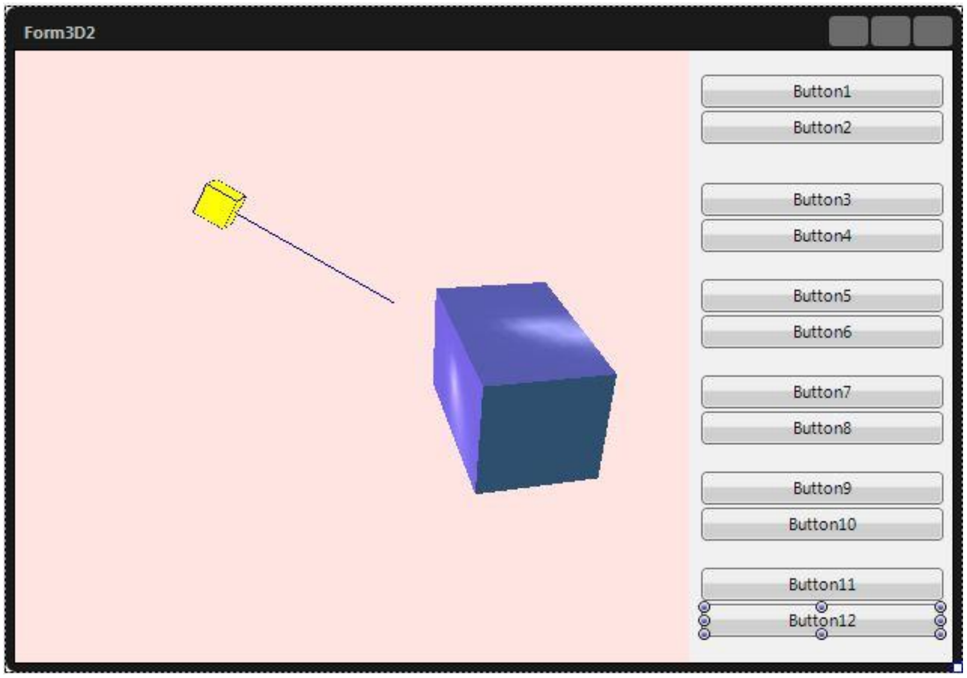
A 3D application cannot directly use HD components such as buttons or lists. However, there is a simple way to deal with this problem, by creating a bridge between the 3D and HD scenes. FireMonkey provides the **TLayer3D** component for this purpose. Use the following steps to add a TLayer3D and HD controls to the 3D project you just built.

Step 1: With the Form selected in either the Object Inspector or the Structure View, double-click TLayer3D in the Tool Palette. The TLayer3D component will appear on the form. Now you can use the TLayer3D component as the surface for FireMonkey HD components.

Step 2: As you can see in the form designer, the TLayer3D component has overshadowed the scene. To ensure visibility of both the stage and the layer that contains your HD controls, use the Object Inspector to set the following properties of the TLayer3D component:

- Set the **Projection** property of the TLayer3D to **pjScreen**. A screen projection is a screen plane pixel-to-pixel projection with a fixed (invisible) camera, and the front plane of the 3d-frustum is now a screen plane. Every object is still 3D, but can be manipulated as if 2D.
- Set the **Align** property to **alRight**
- Set the **Width** property to 175 (we're going to add some TButtons inside the TLayer3D in the next step)

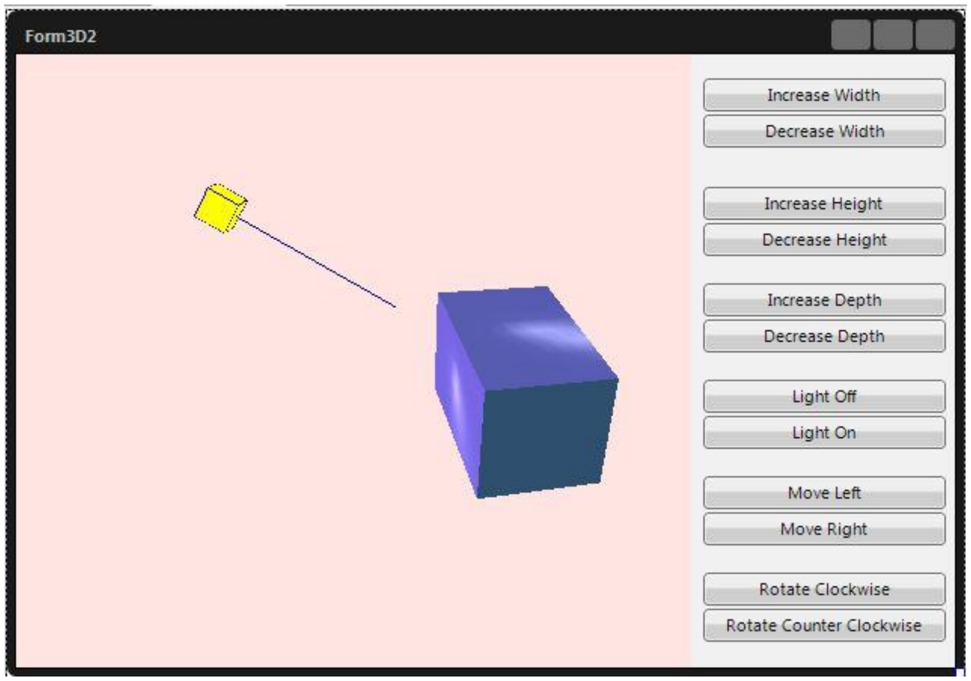
Step 3: With the TLayer3D selected in the Structure View, add 12 TButtons and lay them out in 6 groups of two buttons each. Ten of these buttons are going to be used to control the Height, Width, Depth, Position and rotation of the cube. Two of these buttons are going to be used to control the light. Your form should look something like the following:



Step 4: Change the **Text** and **Name** properties for the button pairs to be:

- Text: Increase Width, Name: IncreaseWidth
- Text: Decrease Width, Name: DecreaseWidth
- Text: Increase Height, Name: IncreaseHeight
- Text: Decrease Height, Name: DecreaseHeight
- Text: Increase Depth, Name: IncreaseDepth
- Text: Decrease Depth, Name: DecreaseDepth
- Text: Light Off, Name: LightOff
- Text: Light On, Name: LightOn
- Text: Move Left, Name: MoveLeft
- Text: Move Right, Name: MoveRight
- Text: Rotate Clockwise, Name: RotateClockwise
- Text: Rotate Counter Clockwise, Name: RotateCounterClockwise

Your form should look like this:



Step 5: Add event handlers for each of the buttons.

- Double click each of the increase and decrease buttons and set the cube's Width, Height and Depth properties to increase and decrease by 1 each event handler.
- Double click each of the Light buttons and set the Enabled property for Light On to True (for Delphi) and 1 (for C++) and for Light Off to False (Delphi) and 0 (for C++).
- Double click each of the Move buttons and for Move Left subtract 1 and for Move Right add 1 to the cube's Position.X property.
- Double click each of the Rotate buttons and for Rotate Clockwise add a value of 10 (degrees) and for Rotate Counter Clockwise subtract a value of 10 (degrees) for the RotationAngle.X property.

Here is the event handler code for Delphi and C++:

```
Delphi Code:

procedure TForm2.IncreaseWidthClick(Sender: TObject);
begin
  Cube1.Width := Cube1.Width + 1
end;

procedure TForm2.DecreaseWidthClick(Sender: TObject);
begin
  Cube1.Width := Cube1.Width - 1;
end;

procedure TForm2.IncreaseHeightClick(Sender: TObject);
begin
  Cube1.Height := Cube1.Height + 1
end;
```

```
procedure TForm2.DecreaseHeightClick(Sender: TObject);
begin
  Cube1.Height := Cube1.Height - 1
end;

procedure TForm2.IncreaseDepthClick(Sender: TObject);
begin
  Cube1.Depth := Cube1.Depth + 1
end;

procedure TForm2.DecreaseDepthClick(Sender: TObject);
begin
  Cube1.Depth := Cube1.Depth - 1
end;

procedure TForm2.LightOffClick(Sender: TObject);
begin
  Light1.Enabled := False
end;

procedure TForm2.LightOnClick(Sender: TObject);
begin
  Light1.Enabled := True
end;

procedure TForm2.MoveLeftClick(Sender: TObject);
begin
  Cube1.Position.X := Cube1.Position.X - 1
end;

procedure TForm2.MoveRightClick(Sender: TObject);
begin
  Cube1.Position.X := Cube1.Position.X + 1
end;

procedure TForm2.RotateClockwiseClick(Sender: TObject);
begin
  Cube1.RotationAngle.X := Cube1.RotationAngle.X + 10
end;

procedure TForm2.RotateCounterClockwiseClick(Sender: TObject);
begin
  Cube1.RotationAngle.X := Cube1.RotationAngle.X -10
end;
```

C++ Code:

```
//---------------------------------------------------------
void __fastcall TForm3D2::IncreaseWidthClick(TObject *Sender)
{
  Cube1->Width = Cube1->Width + 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::DecreaseWidthClick(TObject *Sender)
{
  Cube1->Width = Cube1->Width - 1;
}
```

```
//---------------------------------------------------------
void __fastcall TForm3D2::IncreaseHeightClick(TObject *Sender)
{
  Cube1->Height = Cube1->Height + 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::DecreaseHeightClick(TObject *Sender)
{
  Cube1->Height = Cube1->Height - 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::IncreaseDepthClick(TObject *Sender)
{
     Cube1->Depth = Cube1->Depth + 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::DecreaseDepthClick(TObject *Sender)
{
     Cube1->Depth = Cube1->Depth - 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::LightOffClick(TObject *Sender)
{
  Light1->Enabled = False;
}
//---------------------------------------------------------
void __fastcall TForm3D2::LightOnClick(TObject *Sender)
{
  Light1->Enabled = True;
}
//---------------------------------------------------------

void __fastcall TForm3D2::MoveLeftClick(TObject *Sender)
{
  Cube1->Position->X = Cube1->Position->X - 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::MoveRightClick(TObject *Sender)
{
  Cube1->Position->X = Cube1->Position->X + 1;
}
//---------------------------------------------------------
void __fastcall TForm3D2::RotateClockwiseClick(TObject *Sender)
{
  Cube1->RotationAngle->X = Cube1->RotationAngle->X + 10;
}
//---------------------------------------------------------
void __fastcall TForm3D2::RotateCounterClockwiseClick(
                           TObject *Sender)
{
  Cube1->RotationAngle->X = Cube1->RotationAngle->X - 10;
}
//---------------------------------------------------------
```

Step 6: Run the project by pressing F9. When you click the increase/decrease buttons, the cube's size changes. Turning the light off means that the cube has no light or shadows, so it is entirely dark. Click the Light On button to turn the light back on. When you click the Move Left / Move Right buttons, the cube's position changes. Click the Move Left button. The cube moves to left. When you click the Rotate Clockwise / Counter Clockwise buttons, the cube's rotation angle changes. Click the Rotate Clo ckwise button several times. The cube inclines to right. Observe how the effect of the light changes the shadows on the cube when it inclines.

Adding 3D Cameras, Lights and Cubes to an HD Application

If you have a Windows and Mac HD application you can add 3D to it by using the TViewPort3D component. The following example shows you how to combine HD, 3D and use multiple cameras. You'll learn how to use the TCamera object to define the scene perspective and projection of the objects in the scene to the viewport.

Use the following steps to add 3D components to an HD application.

Step 1: Start the HD project

Choose **File > New > FireMonkey HD Application – Delphi** or **File > New > FireMonkey HD Application – C++Builder** to start your HD application. Use the Form Designer to create two parts of your UI – the Control Panel and the View Port. Set the **Caption** property for the form to "HD and 3D Windows and Mac Application".

Step 2: Building the Control Panel

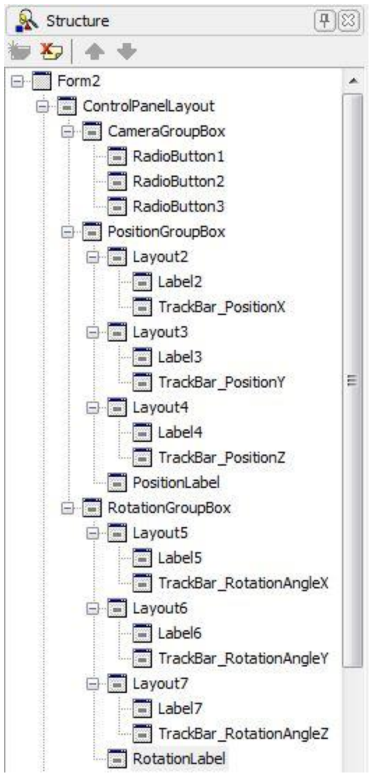
To be able to observe how the scene perspective and projection change, depending on the cameras position and rotation angle, create a control panel that allows changing the properties of the TCamera components.

- In the Tool Palette, search and add a TLayout component. Set the **Align** property to **alRight**. Set the **Width** property to **150**. Set the **Name** property to "ControlPanelLayout".
- Add 3 TGrou pBox components, contained in the TLayout. Set the **Height** property of GroupBox 2 and 3 to *120*. Set each GroupBox's **Width** property to *137*. Evenly space the three group boxes in the Layout area.
- Set the **Text** property of the first TGroup Box to "Camera". Set the **Name** property for the first TGroupBox to "CameraGroupBox". Add 3 TRadioButton components to the GroupBox and set their **Text** properties to "Camera1", "Camera2" and "Camera3".
- Set the **Text** property of the second TGroupBox component to "Position". Set the **Name** property for the second TGroupBox to "PositionGroupBox". Add 3 TLayout components to the second GroupBox. Set the **Height** property of each TLayout to 25. Set the **Width** property of each TLayout to 121. Inside of each TLayout add a TLabel and a TTrackBar. Set each TLabel's **Align** property to **alLeft**. Set the **Text** properties for the TLabels to "X:", "Y:" and "Z:". Set each TrackBar's **Align** propert to **alRight**. Set the **Width** property of each TrackBar to *110*. Set each TrackBar's **Min** property to *-20*. Set each TrackBar's **Max** property to *20*. Set each TrackBar's

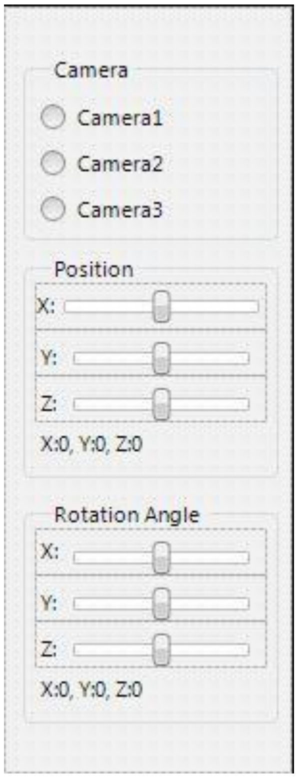
Name property to "*TrackBar_PositionX*", "*TrackBar_PositionY*" and "*TrackBar_PositionZ*" respectively.

- Set the **Text** property of the third TGroupBox component to "RotationAngle". Set the **Name** property for the third TGroupBox to "RotationGroupBox". Add 3 TLayout components to the third GroupBox. Set the **Height** property of each TLayout to 25. Set the **Width** property of each TLayout to **121**. Inside of each TLayout add a TLabel and a TTrackBar. Set each TLabel's **Align** property to **alLeft**. Set the **Text** properties for the TLabels to "X:", "Y:" and "Z:". Set each TrackBar's **Align** property to **alRight**. Set the **Width** property of each TrackBar to **110**. Set each TrackBar's **Min** property to *-90*. Set each TrackBar's **Max** property to *90*. Set each TrackBar's **Name** property to "*TrackBar_RotationAngleX*", "*TrackBar_RotationAngleY*" and "*TrackBar_RotationAngleZ*" respectively.
- To see the values of the TTrackBar objects, add a TLabel component to the bottom of the Position and the RotationAngle group boxes. Set the **Text** property of each TLabel to "X:0, Y:0, Z:0". Set the Label's **Name** properties to "*PositionLabel*" and "*RotationLablel*" respectively.

The Structure View and the Form Designer for the control panel part of the HD/3D application should look something like the following:



```
Form2
  ControlPanelLayout
    CameraGroupBox
      RadioButton1
      RadioButton2
      RadioButton3
    PositionGroupBox
      Layout2
        Label2
        TrackBar_PositionX
      Layout3
        Label3
        TrackBar_PositionY
      Layout4
        Label4
        TrackBar_PositionZ
      PositionLabel
    RotationGroupBox
      Layout5
        Label5
        TrackBar_RotationAngleX
      Layout6
        Label6
        TrackBar_RotationAngleY
      Layout7
        Label7
        TrackBar_RotationAngleZ
      RotationLabel
```



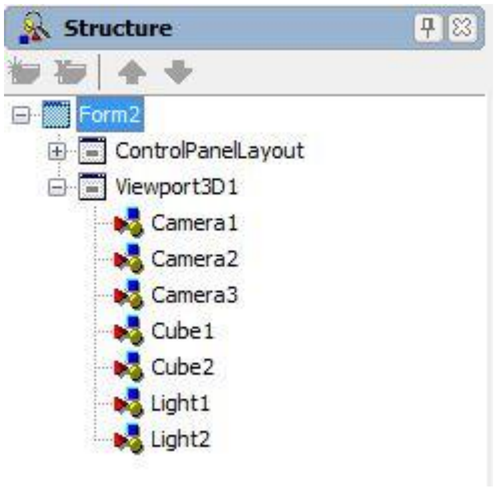
Step 3: Building the View Port

To contain the 3D objects in an HD application you use a TViewPort3D component.

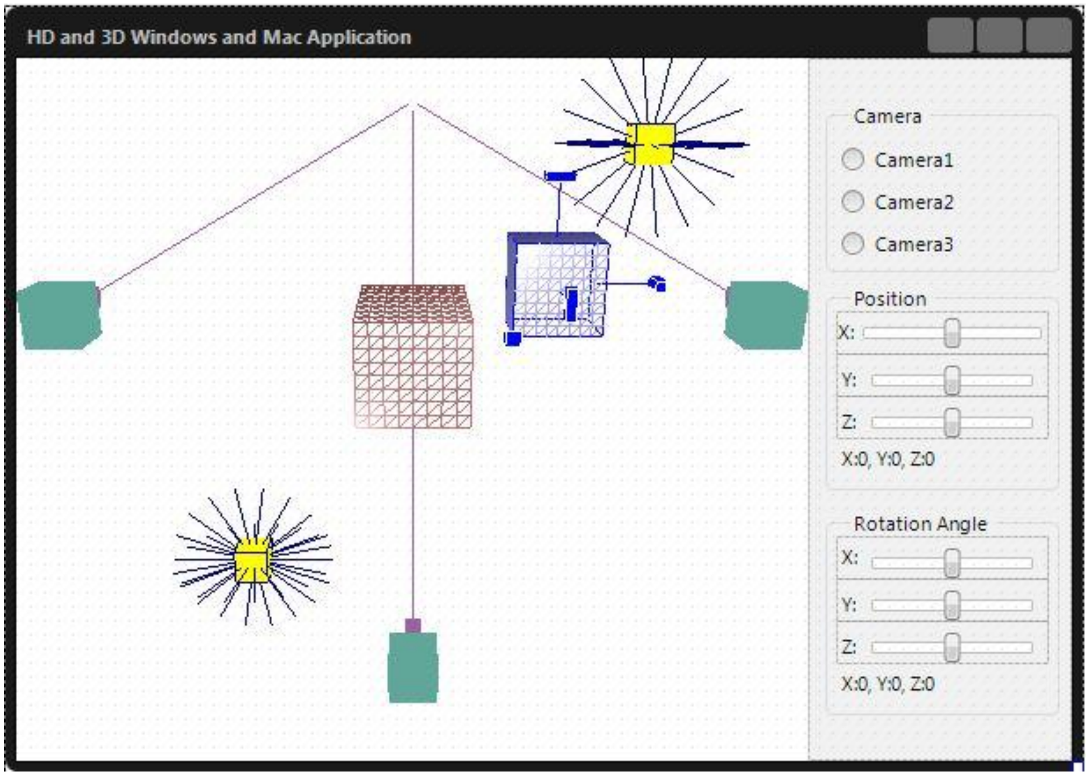
- In the Tool Palette, search and add a TViewport3D component to the form. Set the **Align** property to **alClient**. Set **UseDesignCamera** property to False.
- With the ViewPort3D selected, from the Tool Palette, add the following FireMonkey 3D components:
  - Add 3 TCamera components. Set Camera1's **Position** property to *-5,-4,-7*. Set Camera2's **Position** property to *5,-4,-7*. Set Camera3's **Position** property to *0,2,-7*.
  - Add 2 TCube components. Set Cube1 **Material.Ambient** property to *Red*. Set Cube1 **Material.FillMode** property to *fmWireFrame*. Set Cube1 **Height** property to *5*. Set Cube1 **Width** property to *5*. Set Cube1 **Depth** property to *5*. Set Cube1 **Position** property to *0,0,5*. Set Cube2 **Material.Ambient** property to *Blue*. Set Cube2 **Material.FillMode** property to *fmWireFrame*. . Set Cube2 **Height** property to *3*. Set Cube2 **Width** property to *3*. Set Cube2 **Depth** property to *3*. Set Cube2 **Position** property to *5,-2,5*.
  - Add 2 TLight components. Set both lights **LightType** property to *ltPoint*. Set Light1 Position property to *5,-6,0*. Set Light2 **Position** property to *-4,4,0*.

The TCube objects are the 3D objects that will be seen in the viewport. The TCamera objects will be used to define the position from which the 3D objects are seen. The TLight objects will affect the lighting of the 3D objects, depending on the position of the camera.

After adding the 3D components, your view port Structure View and Form Designer should look something like the following:



```
Form2
  ControlPanelLayout
  Viewport3D1
    Camera1
    Camera2
    Camera3
    Cube1
    Cube2
    Light1
    Light2
```



Step 4: Adding the Event Handlers

For the camera radio buttons, you'll create OnClick events. Add one event handler for all three of the position trackbars **OnChange** event call it "*TrackBar_PositionChange*". Add one event handler for all three of the **OnChange** event call it "*TrackBar_RotationAngleChange*".

```
{ Delphi: for choosing the camera }
procedure TForm2.RadioButton1Click(Sender: TObject);
begin
  CameraChange(Camera1);
end;

procedure TForm2.RadioButton2Click(Sender: TObject);
begin
  CameraChange(Camera2);
end;

procedure TForm2.RadioButton3Click(Sender: TObject);
begin
  CameraChange(Camera3);
end;

// Delphi code for changing the camera position
procedure TForm2.TrackBar_PositionChange(Sender: TObject);
begin
  with Viewport3D1 do begin
    if not FIsChangingCamera then begin
      Camera.Position.X := TrackBar_PositionX.Value;
      Camera.Position.Y := TrackBar_PositionY.Value;
      Camera.Position.Z := TrackBar_PositionZ.Value;
      Repaint;
    end;
    with Camera.Position do begin
      { Updating the displayed values of the coordinates }
      PositionLabel.Text := Format('X: %d, Y: %d, Z: %d',
          [Round(X), Round(Y), Round(Z)]);
    end;
  end;
end;

// Delphi: for changing the rotation angle of the camera
procedure TForm2.TrackBar_RotationAngleChange(Sender: TObject);
begin
  with Viewport3D1 do begin
    if not FIsChangingCamera then begin
      Camera.RotationAngle.X := TrackBar_RotationAngleX.Value;
      Camera.RotationAngle.Y := TrackBar_RotationAngleY.Value;
      Camera.RotationAngle.Z := TrackBar_RotationAngleZ.Value;
      Repaint;
    end;
    with Camera.RotationAngle do begin
      { Updating the displayed values of the coordinates }
      RotationLabel.Text := Format('X: %d, Y: %d, Z: %d',
         [Round(X), Round(Y), Round(Z)]);
    end;
  end;
end;

// C++ code for choosing the camera
void __fastcall TForm2::RadioButton1Click(TObject *Sender)
{
  CameraChange(Camera1);
}
//-----------------------------------------------------------
void __fastcall TForm2::RadioButton2Click(TObject *Sender)
{
  CameraChange(Camera2);
}
//-----------------------------------------------------------
void __fastcall TForm2::RadioButton3Click(TObject *Sender)
{
  CameraChange(Camera3);
}

// C++ code for changing the coordinates of the camera position
void __fastcall TForm2::TrackBar_PositionChange(TObject *Sender)
{
  if (!Form2->FIsChangingCamera) {
    Viewport3D1->Camera->Position->X = TrackBar_PositionX->Value;
    Viewport3D1->Camera->Position->Y = TrackBar_PositionY->Value;
    Viewport3D1->Camera->Position->Z = TrackBar_PositionZ->Value;
    Viewport3D1->Repaint();
  }
  // Update the displayed values of the coordinates
  TVarRec vr[] = {
    round(Viewport3D1->Camera->Position->X),
    round(Viewport3D1->Camera->Position->Y),
    round(Viewport3D1->Camera->Position->Z)
  };
  PositionLabel->Text = Format("X: %d, Y: %d, Z: %d",vr,3);
}

// C++ code for changing the rotation angle of the camera
void __fastcall TForm2::TrackBar_RotationAngleChange(
      TObject *Sender)
{
  if (!Form2->FIsChangingCamera) {
    Viewport3D1->Camera->RotationAngle->X =
      TrackBar_RotationAngleX->Value;
    Viewport3D1->Camera->RotationAngle->Y =
      TrackBar_RotationAngleY->Value;
    Viewport3D1->Camera->RotationAngle->Z =
      TrackBar_RotationAngleZ->Value;
    Viewport3D1->Repaint();
  }
  // Update the displayed values of the coordinates
  TVarRec vr[] = {
    round(Viewport3D1->Camera->RotationAngle->X),
    round(Viewport3D1->Camera->RotationAngle->Y),
    round(Viewport3D1->Camera->RotationAngle->Z)
  };
  RotationLabel->Text = Format("X: %d, Y: %d, Z: %d",vr,3);
}
```

Step 5: Adding a Private Variable and a Private Procedure

Add a variable, *FIsChangingCamera* and *procedure CameraChange* to the form declaration's private section as follows

The private Boolean class variable FIsChangingCamera lets the program know that a button click event handler is updating the camera properties to keep the TrackBar event handlers from also trying to update a camera at the same time.

```
// Delphi code for the form class declaration
private
  { Private declarations }
  FIsChangingCamera : boolean;
  procedure CameraChange(Sender:TObject);

// C++ code for the form class declaration
private:  // User declarations
  bool FIsChangingCamera;
  void CameraChange(TObject *Sender);
```

The CameraChange procedure allows us to provide one camera property change method for all three cameras. The implementation for the CameraChange method:

```
// Delphi code for the CameraChange method implementation
procedure TForm2.CameraChange(Sender: TObject);
begin
```

```
  with Viewport3D1 do begin
    FIsChangingCamera := True;
      try
        Camera := Sender as TCamera;
        TrackBar_PositionX.Value := Camera.Position.X;
        TrackBar_PositionY.Value := Camera.Position.Y;
        TrackBar_PositionZ.Value := Camera.Position.Z;
        Repaint;
      finally
        FIsChangingCamera := False;
      end;
  end;
end;

// C++ code for the CameraChange method implementation
void TForm2::CameraChange(TObject *Sender)
{
  FIsChangingCamera = True;
  try {
    Viewport3D1->Camera = dynamic_cast<TCamera*>(Sender);
    TrackBar_PositionX->Value = Viewport3D1->Camera->Position->X;
    TrackBar_PositionY->Value = Viewport3D1->Camera->Position->Y;
    TrackBar_PositionZ->Value = Viewport3D1->Camera->Position->Z;
    Viewport3D1->Repaint();
  }
  __finally {
    FIsChangingCamera = False;
  }
}
```

For C++ there is also a helper function that implements rounding for the position and rotation labels. Add the following to your main source code file:

```
// C++ code for rounding values for the
//   position and rotation label
int round(float a) {
  if (a < 0)
    return int(a - 0.5);
  else
    return int(a + 0.5);
}
```

Step 6: Run the Application

Hit F9 to run the application and try out the camera selections, position track bars and rotation angle track bars.

Using Fonts in your 3D FireMonkey Applications

You can use Fonts in your 3D applications using HD components or using the TText3D component. The following simple example uses TText3D.

http://docwiki.embarcadero.com/CodeExamples/en/FMXTFont_(Delphi)

[TODO] – Delphi and C++

You can find a complete 3D text example in the RAD Studio sample projects at http://radstudiodemos.svn.sourceforge.net/viewvc/radstudiodemos/trunk/FireMonkey/3DTextEditor/ This example is also covered in a video at http://www.youtube.com/watch?v=VIRQrlKQi2M.

TMesh and TCustomMesh

TMesh represents a 3D wireframe (mesh) that can be used in 3D applications. The TMesh class publishes a set of properties from its ancestor, TCustomMesh, in order to let you use 3D wireframes at design time from within the IDE, through the Object Inspector. You can also create and use TMesh objects at runtime in your code.

TCustomMesh is also used as the base skeleton for other 3D shapes. TCustomMesh is the base class for all non-extruded 3D shapes such as: TCube, TPlane, TDisk, TSphere, TCylinder, TRoundCube and TCone

Anders Ohlsson has written several EDN articles showing you how to use TMesh for 3D Math and Wave visualization using Delphi and C++ for Windows, Mac and iOS.

- Visualizing mathematical functions by generating custom meshes using FireMonkey and Delphi - http://edn.embarcadero.com/article/42007
- Visualizing mathematical functions by generating custom meshes using FireMonkey and C++ - http://edn.embarcadero.com/article/42114
- Visualizing wave interference using FireMonkey and Delphi XE2 - http://edn.embarcadero.com/article/42012
- Visualizing wave interference using FireMonkey and C++Builder XE2 - http://edn.embarcadero.com/article/42115

The source code for each project is available on EDN in CodeCentral. Use the links included in each article.

There is also a Delphi example that shows you how to import 3D Studio (3DS) files into a TMesh at C:\Users\Public\Documents\RAD Studio\9.0\Samples\FireMonkey\import3ds.

TModel3D

TModel3D is a component included in RAD Studio for loading several industry standard 3D model formats. The **MeshCollection** property in TModel3D supports a generic 3D model consisting of a collection of wireframes.

TModel3D component to load one of the three supported 3D model formats: DAE, OBJ and ASE.

- DAE – COLLADA 3D model file format
- OBJ – Wavefront 3D model file format
- ASE – ASCII Scene Export 3D model file in ASCII text format

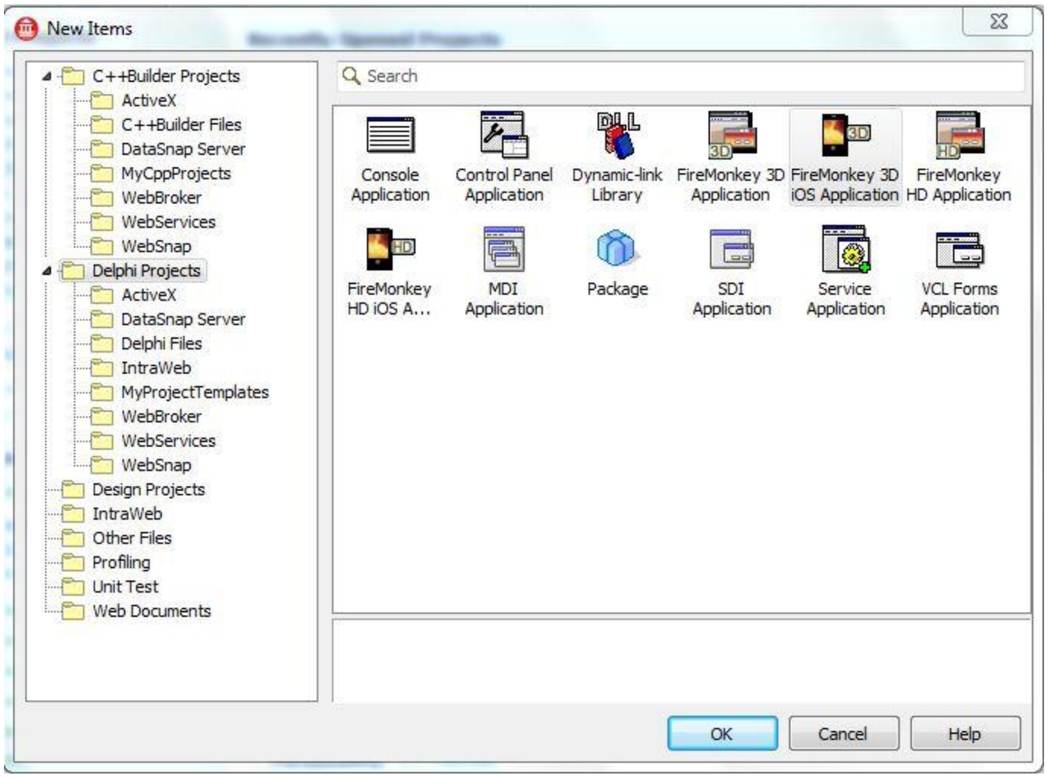
You can find the steps to building a Delphi or C++ 3D TModel3D application on the Embarcadero DocWiki at:

- http://docwiki.embarcadero.com/RADStudio/en/Importing_a_3D_Model_in_a_FireMonkey_Application
- http://docwiki.embarcadero.com/RADStudio/XE2/en/Applying_Simple_Customizations_to_an_Imported_3D_Model

There is also a Delphi example model viewer application that uses TModel3D in your installation samples directory at C:\Users\Public\Documents\RAD Studio\9.0\Samples\FireMonkey\COLLADAModelViewer\projects\Model Viewer\src

Creating a FireMonkey 3D Application for iOS (Delphi only in XE2)

The steps for creating a 3D iOS application are the same as building a Delphi 3D application. The only difference is how you start the project. For 3D iOS use **File > New > Other > Delphi Projects > FireMonkey 3D iOS Application**.



Then follow the steps to building any of the previous Delphi 3D applications. Rember to use the "Export to Xcode" tool menu item after you have created your 3D iOS project. You can run and test your 3D application on Windows. When it is working you can then switch to Xcode on a Mac to build, test and deploy your 3D iOS application to your iPhone, iPad or iPod Touch.

Summary, Looking Forward, To Do Items, Resources, Q&A and the Quiz

In Lesson 7 you learned how to create 3D applications for Windows, Mac and iOS (Delphi only). You learned about the various components including TCamera, TLight, TMesh and TModel3D to create great looking 3D applications and HD applications containing 3D components using TViewPort3D.

In Lesson 8, you'll learn how to use FireMonkey's Effects and Animations components to add visual richness to your applications.

In the meantime, here are some things to do, articles to read and videos to watch to enhance what you learned in Lesson 7 and to prepare you for lesson 8.

To Do Items

Explore the 3D example applications that are included with RAD Studio. Take a look at Anders Ohlsson's FireMonkey 3D articles for Math Visualization and Wave Interaction.

Links to Additional Resources

- Getting Started Course landing page - http://www.embarcadero.com/firemonkey/firemonkey-e-learning-series
- FireMonkey Application Platform - http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_Application_Platform
- FireMonkey 3D Primer - http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_3D

Delphi:

- RAD Studio Delphi sample programs on SourceForge - http://radstudiodemos.svn.sourceforge.net/viewvc/radstudiodemos/branches/RadStudio_XE2/FireMonkey/
- Using Delphi to build FireMonkey 3D Text Editor for Windows and Mac - http://www.youtube.com/watch?v=VIRQrlKQi2M
- FireMonkey 3D FireFlow C++ demo - C:\Users\Public\Documents\RAD Studio\9.0\Samples\FireMonkey\FireFlow
- FireMonkey 3D Animation C++ demo - C:\Users\Public\Documents\RAD Studio\9.0\Samples\FireMonkey\AnimationDemo3
- 3D Text Demo -

- BizFlow – FireMonkey 3D Record View - http://www.youtube.com/watch?v=TbIgGRWGA-I
- FireMonkey 3D application with shaders "FireMonkey Dathox Demo" - http://edn.embarcadero.com/article/41874
- Visualizing physics using FireMonkey - http://edn.embarcadero.com/article/42020
- Creating 3D scenes dynamically in FireMonkey - http://members.adug.org.au/2012/01/02/creating-3d-scenes-dynamically-in-firemonkey/

C++:

- RAD Studio C++ sample programs on SourceForge - http://radstudiodemos.svn.sourceforge.net/viewvc/radstudiodemos/branches/RadStudio_XE2/CPP/FireMonkey/
- FireMonkey 3D FireFlow C++ demo - C:\Users\Public\Documents\RAD Studio\9.0\Samples\CPP\FireMonkey\FireFlow
- FireMonkey 3D Animation C++ demo - C:\Users\Public\Documents\RAD Studio\9.0\Samples\CPP\FireMonkey\AnimationDemo3D

Q&A:

Here are some of the answers for the questions I've received (so far) for this lesson. I will continue to update this Course Book during and after course.

Q:
A:

If you have any additional questions – send me an email - davidi@embarcadero.com

Self Check Quiz

1. Which of the following components is not a 3D component?

a) TSphere
b) TEdit
c) TPath3D
d) TMesh
e) TModel3D

2. Which 3D component allows you to change the user's view of a 3D form ?

a) TLayout3D
b) TViewPort3D
c) TCamera
d) TMesh

3. What values for Position.Z brings 3D objects closer to the user?

a) Z set to zero
b) Z set less than zero
c) Z set greater than zero

Answers to the Self Check Quiz:

1b, 2c, 3b