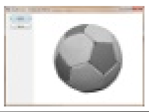
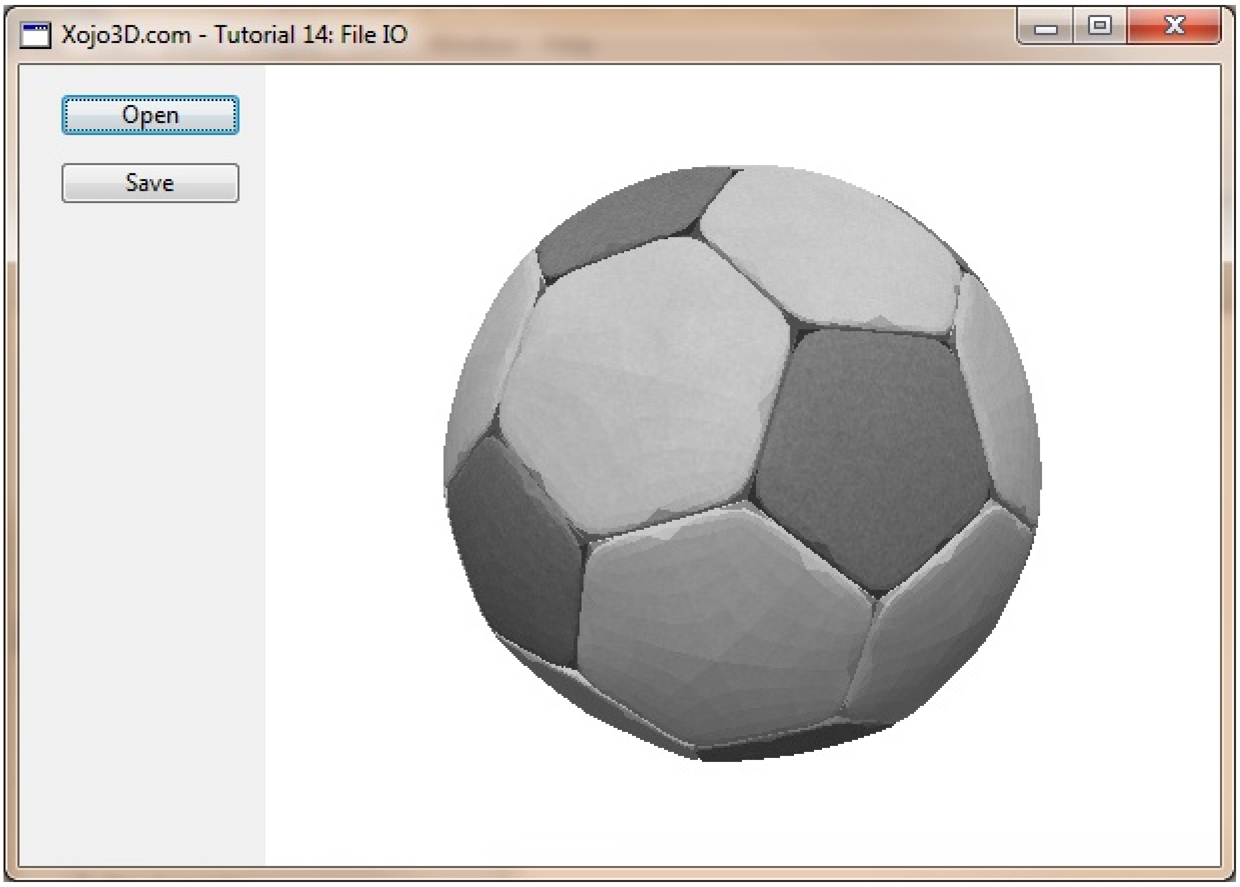


# Tutorial 14: File IO

Saving 3D assets to disk, or transporting them across a network, is an important feature of any professional 3D application. The X3 Data Format is a simple open specification that can be used save and transport 3D assets. This tutorial shows you how to save and load 3D models using the X3 Data Format.



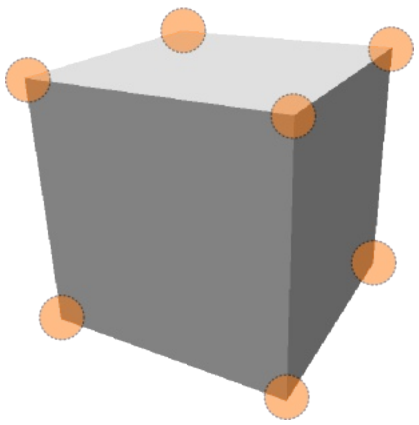
## Theory

The X3 Data Format is a data specification designed for the storage and transmission of 3D assets. The lightweight and basic structure of JSON makes it super easy to work with X3 assets.

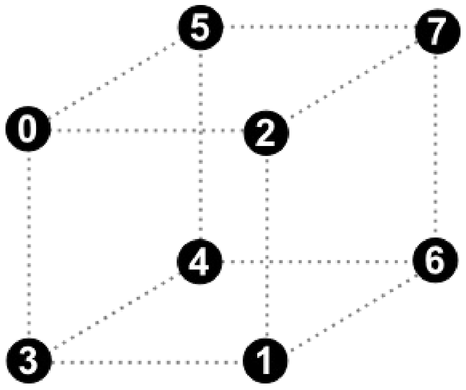
The best way to illustrate the workings of the X3 Data Format is with an example. Below is the data of a basic cube:

```
{
  "x3model":{
    "vertex":[
      -1, 1, 1,
      1, -1, 1,
      1, 1, 1,
      -1, -1, 1,
      -1, -1, -1,
      -1, 1, -1,
      1, -1, -1,
      1, 1, -1
    ],
    "polygon":[
      {"vi":[0, 3, 1, 2]},
      {"vi":[0, 5, 4, 3]},
      {"vi":[2, 1, 6, 7]},
      {"vi":[2, 7, 5, 0]},
      {"vi":[1, 3, 4, 6]},
      {"vi":[7, 6, 4, 5]}
    ]
  }
}
```

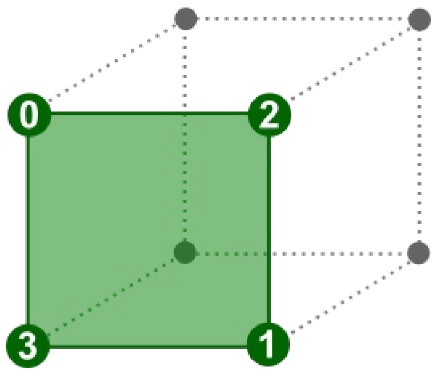
The **x3model** object indicates that this is an X3 model asset. The object contains all the information needed to render the model, such as geometrical data, colors, textures, etc. The image to the right illustrates what the cube looks like. The orange spots indicate where the vetices are located.



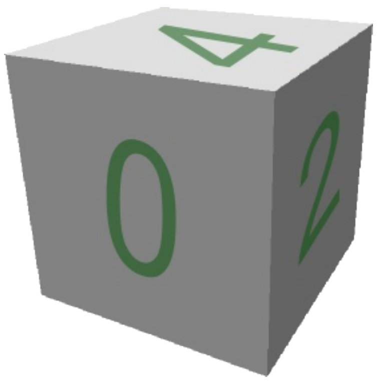
The first child of x3model is the **vertex** object. This object stores all the 3D vertices used by the model, as a one-dimensional numerical array. The first three values in the array is the respective X, Y and Z values of the first vertex. The next three values in the array is the respective X, Y and Z values of the second vertex, and so forth ... This cube model has eight vertices stored in its global vertex array.



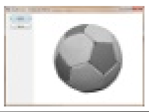
The second child of x3model, **polygon**, stores all the polygon objects of the model. Each polygon has different attributes such as color or texture, but in this cube example we only define the geometrical shape of the polygon by making use of the vindex array. When no color or texture is set for the polygon, the polygon will automatically be rendered using a white color.



The image below illustrates some of the surfaces together with their respective index in the polygon array.



You can download the latest X3 Data Format Specification from www.Xojo3D.com for the full list of features provided by the X3 Data Format.

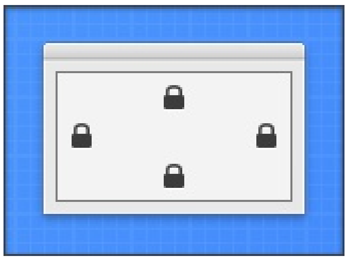


## Tutorial Steps

1. Create a new Xojo desktop project.
2. Save your project.
3. Import the X3Core module.
   You can download the module from http://www.xojo3d.com/tutorials/tut014/x3core.zip.
4. Configure the following controls:

| Control | Name | Left | Top | Caption | DoubleBuffer | Maximize Button |
|---|---|---|---|---|---|---|
| Window | SurfaceWindow | - | - | - | - | ON |
| OpenGLSurface | Surface | 123 | 0 | - | ON | - |
| Generic Button | OpenButton | 20 | 14 | Open | - | - |
| Generic Button | SaveButton | 20 | 48 | Save | - | - |

5. Position and size *Surface* to fill the whole part of the window not occupied by the buttons, and set its locking to left, top, bottom and right.



6. Add the following code to the *SurfaceWindow.Open* event handler:

```
Self.MouseCursor = System.Cursors.StandardPointer
```

7. Add the following code to the *SurfaceWindow.Paint* event handler:

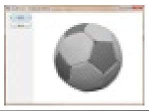
```
Surface.Render
```

8. Add the following code to the *Surface.Open* event handler:

```
X3_Initialize

X3_EnableLight OpenGL.GL_LIGHT0, new X3Core.X3Light(0, 1, 1)
```

9. Add the following code to the *Surface.Resized* event handler:

```
X3_SetPerspective Surface
```



10. Add the following properties to *SurfaceWindow*:

| Name | Type |
|---|---|
| Model | X3Core.X3Model |
| MousePrevX | Integer |
| MousePrevY | Integer |

11. Add the following code to the *Surface.Render* event handler:

```
OpenGL.glClearColor(1, 1, 1, 1)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT +
                  OpenGL.GL_DEPTH_BUFFER_BIT)

OpenGL.glPushMatrix

OpenGL.glTranslatef 0, 0, -2.5

if Model <> nil then
  X3_RenderModel Model
end if

OpenGL.glPopMatrix
```

12. Add the following code to the *Surface.MouseDown* event handler:

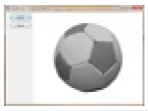
```
MousePrevX = x
MousePrevY = y

return true
```

13. Add the following code to the *Surface.MouseDrag* event handler:

```
X3_RotateWithXY Model.Rotation, (y - MousePrevY), (x - MousePrevX)

Surface.Render

MousePrevX = x
MousePrevY = y
```

14. Add the following code to the *OpenButton.Action* event handler:

```
Dim dlg As new OpenDialog
Dim modFile As FolderItem
Dim x3mType As new FileType
Dim io as new X3Core.X3IO

// continue on next page
```



```
// continued from previous page

x3mType.Name = "X3 Model"
x3mType.Extensions = "x3m"

dlg.Filter = x3mType

modFile = dlg.ShowModal()

if modFile <> nil then
  Model = io.LoadModel(modFile)
  Surface.Render
end if
```

15. Add the following code to the SaveButton.Action event handler:

```
Dim dlg As new SaveAsDialog
Dim modFile As FolderItem
Dim x3mType As new FileType
Dim io as new X3Core.X3IO

x3mType.Name = "X3 Model"
x3mType.Extensions = "x3m"

dlg.Filter = x3mType

modFile = dlg.ShowModal()

if modFile <> nil then
  if Right(modFile.Name, 4) <> ".x3m" then
    modFile.Name = modFile.Name + ".x3m"
  end if
  io.SaveModel(modFile, Model)
end if
```

16. Download and extract the test model.

You can download the test model from http://www.xojo3d.com/tutorials/tut014/ball.zip.

17. Save and run your project. Open the test file with the open button.

## Analysis

The X3IO module contains the required methods to respectively load and save X3 assets, to and from files.

**OpenButton.Action:**

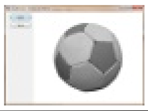
```
Dim dlg As new OpenDialog
Dim modFile As FolderItem
Dim x3mType As new FileType
Dim io as new X3Core.X3IO

x3mType.Name = "X3 Model"
x3mType.Extensions = "x3m"

dlg.Filter = x3mType

modFile = dlg.ShowModal()

if modFile <> nil then
  Model = io.LoadModel(modFile)
  Surface.Render
end if
```

X3 models are stored in files with the extension **.x3m**. In the code above the x3mType object defines this extension, and is used together with an OpenDialog object to select a model to load.

To load an X3 model from a file we instantiate a new X3IO object, and call its LoadModel method with a FolderItem as a parameter that points to the file. The result is then assigned to your model object.

**OpenButton.Action:**

```
Dim dlg As new SaveAsDialog
Dim modFile As FolderItem
Dim x3mType As new FileType
Dim io as new X3Core.X3IO

x3mType.Name = "X3 Model"
x3mType.Extensions = "x3m"

dlg.Filter = x3mType

modFile = dlg.ShowModal()
// continue on next page
```



```
if modFile <> nil then
  if Right(modFile.Name, 4) <> ".x3m" then
    modFile.Name = modFile.Name + ".x3m"
  end if
  io.SaveModel(modFile, Model)
end if
```

Saving an X3 model is just as easy as loading a model. Simply call the SaveModel method of the X3IO object, together with a FolderItem and the model to be saved.