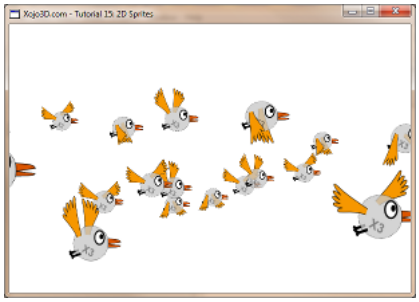


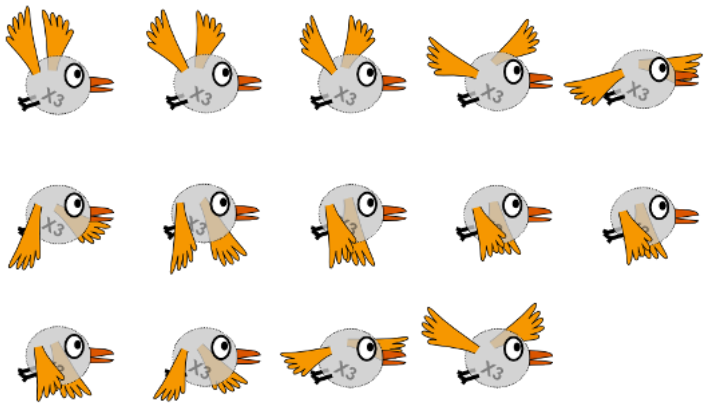
Tutorial 15: 2D Sprites

Sprites provide an efficient way to perform 2D animations. In this tutorial you will learn all about sprites, sprite sheets and how to implement them with OpenGL, while still having full access to all your 3D routines written in the previous tutorials.

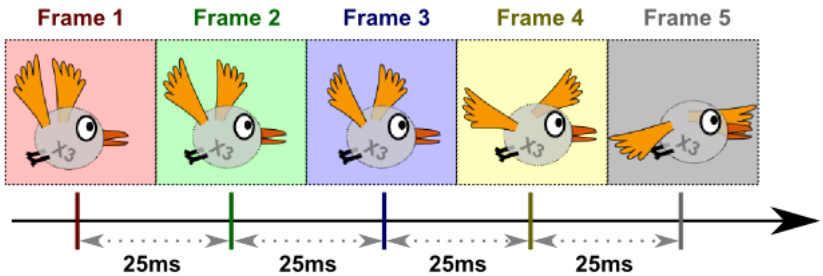
Windows Binary OS X Binary Linux Binary (32 bit)
Download Tutorial PDF Download Xojo Project

Theory

A **sprite** is an animation with its frames stored as a single bitmap, better known as a **sprite sheet** or a **tile sheet**. Below is a sprite sheet that might b animation of a flying bird.



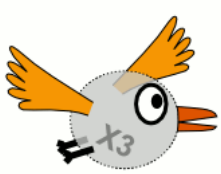
An animation is created from a sprite sheet by showing one frame at a time with a short pause between each frame. The duration of the pause will de you want to achieve with the sprite. The animation sequence might be illustrated as follow:



The sprite animation applies the following logic:

- Render frame 1
- Wait 25 milliseconds
- Render frame 2
- Wait 25 milliseconds
- ...
- Render frame n
- Wait 25 milliseconds
- Go back to frame 1

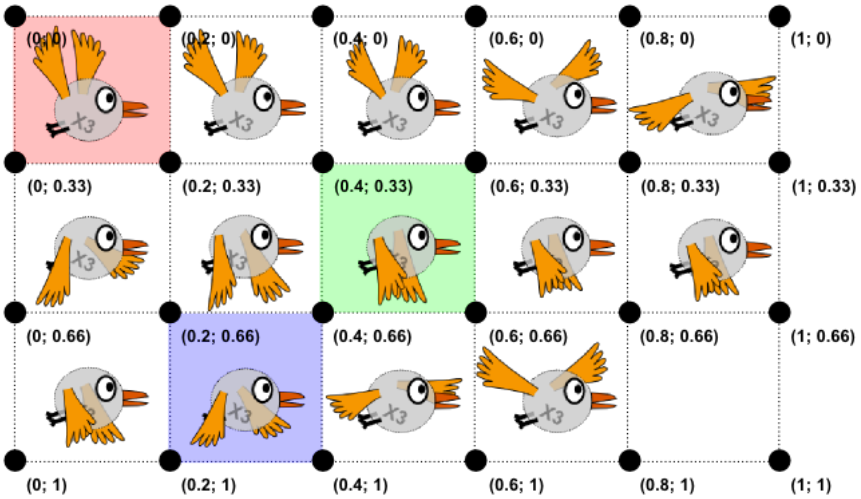
The final result of the animation sequence might looks as follow...



But how do we add 2D sprites to our 3D application, and still have access to all our 3D routines?... by applying the same technique we did with textu "painting" the sprite onto a polygon.

The only difference between a texture and a sprite, is that a sprite is a texture whose image constantly changes. Once a polygon has been "sprite ma apply the same rotation, translation and scaling operations to the polygon, like we would with any other polygon.

An easy way to implement a sprite with OpenGL is to load the sprite sheet like you would load a normal texture, and then to define the different UV c frames.



The UV Maps for the red, green and blue frames in the sprite sheet above are as follow:

Red	[0, 0, 0, 0.33, 0.2, 0.33, 0.2, 0]
Green	[0.4, 0.33, 0.4, 0.66, 0.6, 0.66, 0.6, 0.33]
Blue	[0.2, 0.66, 0.2, 1, 0.4, 1, 0.4, 0.66]

Once we have all the UV coordinates defined for the sprite sheet, we compile a list of UV coordinates for each frame and store these lists in an array only store the top-left coordinate and bottom-right coordinate because the other two coordinates can be determined from these two.

Frame 1	[0, 0, 0.2, 0.33]
Frame 2	[0.2, 0, 0.4, 0.33]
	...
Frame 14	[0.6, 0.66, 0.8, 1]

To animate a "sprite mapped" polygon we simply bind to the sprite sheet texture, and set the UV-map coordinates of the polygon to the coordinates c frame. With a timer we then change the UV-map coordinates of the polygon to the coordinates of the second frame after a short delay, and redraw th new coordinates. We keep on doing this until we reach the last frame, at which point we can switch back to the first frame.

Tutorial Steps

1. Create a new Xojo desktop project.
2. Save your project.
3. Import the X3Core module.
4. Configure the following controls:

Control	Name	DoubleBuffer	Left	Top	Maximize Button	Period
Window	SurfaceWindow	-	-	-	ON	-
OpenGLSurface	Surface	ON	0	0	-	-
Timer	tmrAnimate	-	-	-	-	20

5. Position and size *Surface* to fill the window, and set its locking to left, top, bottom and right.

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

```
Surface.Render
```

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

```
X3_Initialize
X3_EnableLight OpenGL.GL_LIGHT0, new X3Core.X3Light(0, 1, 1)
```

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

```
X3_SetPerspective Surface
```

9. Add a new class named "X3SpriteCoordinate" to module *X3Core*.
10. Add the following properties to *X3SpriteCoordinate*:

Name	Type
U1	Double
V1	Double
U2	Double
V2	Double

11. Add the following method to *X3SpriteCoordinate*:

```
Sub Constructor(initU1 As Double, initV1 As Double, initU2 As Double, initV2 As Double)
  U1 = initU1
  V1 = initV1
  U2 = initU2
  V2 = initV2
End Sub
```

12. Add the following method to *X3SpriteCoordinate*:

```
Function Clone() As X3Core.X3SpriteCoordinate
  Dim coord As new X3SpriteCoordinate(U1, V1, U2, V2)
  return coord
End Function
```

13. Add a new class named "X3Sprite" to module *X3Core*.
14. Add the following properties to *X3Sprite*:

Name	Type
SpriteSheet	X3Core.X3Texture
SpriteMap()	X3Core.X3SpriteCoordinate

15. Add the following method to *X3Sprite*:

```
Sub Constructor()
  ' nothing to do
End Sub
```

16. Add the following method to *X3Sprite*:

```
Sub Constructor(initSheet As Picture)
  SpriteSheet = new X3Core.X3Texture(initSheet)
End Sub
```

17. Add the following method to *X3Sprite*:

```
Sub Constructor(initSheet As Picture, spriteWidth As Integer, spriteHeight As Integer, imageCount As Integer)
  Dim i As Integer
  Dim colCount As Integer
  Dim row As Integer
  Dim col As Integer
  Dim sc As X3Core.X3SpriteCoordinate
  Dim u1 As Double
  Dim v1 As Double
  Dim u2 As Double
  Dim v2 As Double

  SpriteSheet = new X3Core.X3Texture(initSheet)

  colCount = spriteSheet.Width \ spriteWidth

  i = 0
  while i < imageCount

    row = i \ colCount
```

```
    col = i mod colCount

    u1 = round((col * spriteWidth) / spriteSheet.Width * 100) / 100
    v1 = round((row * spriteHeight) / spriteSheet.Height * 100) / 100
    u2 = round(((col + 1) * spriteWidth) / spriteSheet.Width * 100) / 100
    v2 = round(((row + 1) * spriteHeight) / spriteSheet.Height * 100) / 100

    sc = new X3Core.X3SpriteCoordinate(u1, v1, u2, v2)

    SpriteMap.Append sc

    i = i + 1
  wend
End Sub
```

18. Add the following method to *X3Sprite*:

```
Function Clone() As X3Core.X3Sprite
  Dim sprite As New X3Core.X3Sprite
  Dim i As Integer

  for i = 0 to SpriteMap.Ubound
    sprite.SpriteMap.Append SpriteMap(i).Clone()
  next i

  sprite.SpriteSheet = SpriteSheet

  return sprite
End Function
```

19. Add the following properties to *X3Model*:

Name	Type
Sprite()	X3Core.X3Sprite
SpritePolygons()	X3Core.X3Polygon

20. Add the following properties to *X3Polygon*:

Name	Type	Default
SIndex	Integer	-1
SpriteImageCount	Integer	
SpriteImageIndex	Integer	

21. Convert *SpriteImageIndex* into a computed property.
22. Change the default value of *mSpriteImageIndex* to -1.
23. Change the Set method of the *SpriteImageIndex* computed property to:

```
Dim model As X3Core.X3Model
Dim s As X3Core.X3Sprite
Dim uvc As X3Core.X3UVCoordinate
Dim sc As X3Core.X3SpriteCoordinate

model = ParentModel

if SIndex >= 0 then

  s = model.Sprite(SIndex)

  mSpriteImageIndex = value

  if (mSpriteImageIndex <= s.SpriteMap.Ubound) and (mSpriteImageIndex >= 0) then

    sc = s.SpriteMap(mSpriteImageIndex)

    uvc = model.UVMap(UVIndex(0))
    uvc.U = sc.U1
    uvc.V = sc.V1

    uvc = model.UVMap(UVIndex(1))
    uvc.U = sc.U1
    uvc.V = sc.V2

    uvc = model.UVMap(UVIndex(2))
    uvc.U = sc.U2
    uvc.V = sc.V2

    uvc = model.UVMap(UVIndex(3))
    uvc.U = sc.U2
    uvc.V = sc.V1

  end if

end if
```

24. Convert *SpriteImageCount* into a computed property.
25. Remove the *mSpriteImageCount* property that was created by the previous step.
26. Remove all the code from the *SpriteImageCount* Set method to effectively make it a read-only property.
27. Change the Get method of the *SpriteImageCount* computed property to:

```
Dim cnt As Integer
Dim s As X3Core.X3Sprite

cnt = 0

if SIndex >= 0 then
  s = ParentModel.Sprite(SIndex)
  cnt = s.SpriteMap.Ubound + 1
end if

return cnt
```

28. Download spritesheet.png and save it next to your project file.
29. Import the picture into your project and rename it to "imgSpritesheet".
30. Add the following properties to *SurfaceWindow*:

Name	Type
Sprite()	X3Core.X3Model
SpriteStep()	Double

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

```
Dim masterSprite As X3Core.X3Model
Dim spriteModel As X3Core.X3Model
Dim rnd As new Random()
Dim i As Integer
Dim j As Integer
Dim tmpMod As X3Core.X3Model

Self.MouseCursor = System.Cursors.StandardPointer

masterSprite = X3_CreateSprite(imgSpritesheet, 182, 169, 14)

for i = 1 to 20

  spriteModel = masterSprite.Clone()

  spriteModel.Position.X = -rnd.InRange(50, 200) / 10
  spriteModel.Position.Y = (rnd.InRange(0, 40) - 20) / 10
  spriteModel.Position.Z = -rnd.InRange(100, 900) / 100

  spriteModel.Polygon(0).SpriteImageIndex = rnd.InRange(0, spriteModel.Polygon(0).SpriteImageCount - 1)

  Sprite.Append spriteModel

  SpriteStep.Append rnd.InRange(80, 160) / 1000
next i

for j = 0 to Sprite.Ubound
  for i = 0 to Sprite.Ubound - 1
    if Sprite(i).Position.Z > (Sprite(i + 1).Position. Z) then
      tmpMod = Sprite(i)
      Sprite(i) = Sprite(i + 1)
      Sprite(i + 1) = tmpMod
    end if
  next i
next j
```

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

```
Dim i As Integer

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

for i = 0 to Sprite.Ubound
  X3_RenderModel Sprite(i)
next i

OpenGL.glPopMatrix
```

33. Add the following code to the *tmrAnimate.Action* event handler:

```
Dim i As Integer

for i = 0 to Sprite.Ubound

  Sprite(i).NextSpriteImage()
  Sprite(i).Position.X = Sprite(i).Position.X + SpriteStep(i)

next i

Surface.Render
```

34. Save and run your project.

Analysis

The new X3Sprite class is central to the rendering of our sprites. Let's have a closer look at this new class.

X3Sprite.Constructor:

```
Sub Constructor(initSheet As Picture, spriteWidth As Integer, spriteHeight As Integer, imageCount As Integer)
  Dim i As Integer
  Dim colCount As Integer
  Dim row As Integer
  Dim col As Integer
  Dim sc As X3Core.X3SpriteCoordinate
  Dim u1 As Double
  Dim v1 As Double
  Dim u2 As Double
  Dim v2 As Double

  SpriteSheet = new X3Core.X3Texture(initSheet)

  colCount = spriteSheet.Width \ spriteWidth

  i = 0
  while i < imageCount

    row = i \ colCount
    col = i mod colCount

    u1 = round((col * spriteWidth) / spriteSheet.Width * 100) / 100
    v1 = round((row * spriteHeight) / spriteSheet.Height * 100) / 100
    u2 = round(((col + 1) * spriteWidth) / spriteSheet.Width * 100) / 100
    v2 = round(((row + 1) * spriteHeight) / spriteSheet.Height * 100) / 100

    sc = new X3Core.X3SpriteCoordinate(u1, v1, u2, v2)

    SpriteMap.Append sc

    i = i + 1
  wend
End Sub
```

The above constructor initializes a new sprite object. The given parameters are used to initialize a new texture from the given sprite sheet p
while the width, height and image count parameters are used to determine the UV points on the sprite map. These UV points are then use
instantiate an X3SpriteCoordinate object for each frame.

The newly created texture and array of X3SpriteCoordinate objects contain all the information we need to loop through the different frames
sprite during our animation sequences.

X3SpriteCoordinate.Constructor:

```
Sub Constructor(initU1 As Double, initV1 As Double, initU2 As Double, initV2 As Double)
  U1 = initU1
  V1 = initV1
  U2 = initU2
  V2 = initV2
End Sub
```

An X3SpriteCoordinate class is initialized with four values, namely U1, V1, U2 and V2. These are UV points on the sprite sheet, with (U1,V
the top-left corner of a sprite frame, and (U2,V2) being the bottom-right corner. It is, therefore, easy to see that the two coordinates defines
that a single sprite frame occupies on a sprite sheet.

SurfaceWindow.Open:

```
Dim masterSprite As X3Core.X3Model
Dim spriteModel As X3Core.X3Model
Dim rnd As new Random()
Dim i As Integer
Dim j As Integer
```

```
Dim tmpMod As X3Core.X3Model

Self.MouseCursor = System.Cursors.StandardPointer

masterSprite = X3_CreateSprite(imgSpritesheet, 182, 169, 22)

for i = 1 to 20

  spriteModel = masterSprite.Clone()

  spriteModel.Position.X = -rnd.InRange(50, 200) / 10
  spriteModel.Position.Y = (rnd.InRange(0, 30) - 15) / 10
  spriteModel.Position.Z = -rnd.InRange(100, 900) / 100

  spriteModel.Polygon(0).SpriteImageIndex = rnd.InRange(0, spriteModel.Polygon(0).SpriteImageCount - 1)

  Sprite.Append spriteModel

  SpriteStep.Append rnd.InRange(80, 160) / 1000
next i

for j = 0 to Sprite.Ubound
  for i = 0 to Sprite.Ubound - 1
    if Sprite(i).Position.Z > (Sprite(i + 1).Position. Z) then
      tmpMod = Sprite(i)
      Sprite(i) = Sprite(i + 1)
      Sprite(i + 1) = tmpMod
    end if
  next i
next j
```

The above code is where we put our new classes, and the changes to the existing classes, to good use.

First we create a "master" sprite. It is from this sprite object that we will clone many other sprites. The reason that we do this, is because it clone a new sprite from an existing sprite, than it is to initialize a new sprite using the sprite sheet each time. Cloned sprites also share the sprite sheet texture, thereby improving memory usage.

The X3_CreateSprite helper function makes it super easy to instantiate a new sprite, by simply passing the sprite sheet in the form of a nor picture object, the width and height of the frames, and the number of frames in the sprite sheet, as parameters.

Once the master sprite is initialized, we create 20 clone sprites. In the for loop the logic is as follow:

- Create a clone from the master sprite.
- Give the sprite a random position in 3D space.
- Then give the sprite a random starting position by changing the polygon's sprite image index to a random value. (Remember, a sprite a texture mapped onto polygons, whose image constantly changes.)
- Store the new sprite in our array of sprite models.
- Generate a random step for the sprite and store it in our sprite step array.

When working with transparent objects, and most sprites have transparent parts, one challenge is to render the models farthest from the ca and models closest to the camera last. Luckily with 2D our camera position doesn't change so we can simply sort our objects once in our s routine. In the above code this is achieved with a simple bubble sort according to the Z-positions of the models.

tmrAnimate.Action:

```
Dim i As Integer

for i = 0 to Sprite.Ubound

  Sprite(i).NextSpriteImage()
  Sprite(i).Position.X = Sprite(i).Position.X + SpriteStep(i)

next i

Surface.Render
```

It is in the action event of our timer where the animation magic happens. Once our sprite models are initialized, all we have to do is to loop the models and call their NextSpriteImage method to advance the sprites to their next frames. That is all there is to it. The NextSpriteImage will automatically loop back to the start when needed.

We added a second part to the animation by changing the X position of the sprite model each time the sprite frame is updated. This effect illusion of forward moving flight.