orxSHADER structure

Summary

```
[ShaderTemplate]
Code = "// Shader code
  void main()
  {
   // Do stuff
  }"
KeepInCache    = <bool>
ParamList      = ParamFloat # ParamTexture # ParamVector
ParamFloat     = <float>
ParamVector    = <vector>
ParamTexture   = path/to/TextureFile|screen
UseCustomParam = <bool>
```

Details

Here's a list of the available properties for an `orxSHADER` structure:

- `Code`: This block [1] contains the code that will be executed. It needs to be provided and be valid GLSL fragment shader code.
- `KeepInCache`: Defines if the shader code should be kept in memory even if no shader of this type is currently in use. This saves time (reading from disk + compiling) but costs memory. Its default value is `false`.
- ParamList: This defines the list of parameters needed and used by the shader's code. Every defined parameter must have a default value that will help orx guess their type. If none is provided, then its type will be assumed to be a texture. Available types are <float>, <vector> and texture (if a path to a texture file or the keyword `screen` is provided). If an invalid path is provided for a parameter, or the parameter isn't defined at all, the owner's texture will be used [2]. **If an explicit list is provided for any parameter, the shader variable will be an array of this parameter type (instead of a regular variable) and its size will be the number of items in the list.**
- `UseCustomParam`: Defines if parameters can have their value overridden at runtime (ie. interactive). Its default value is `false` which means only the default values will be used.

Here's a simple example of a non-interactive shader as seen in the spawner/shader tutorial.

```
[Decompose]
Code = "void main()
{
  float fRed, fGreen, fBlue;

  // Computes positions with offsets
  vec2 vRedPos    = vec2(gl_TexCoord[0].x + offset.x, gl_TexCoord[0].y +
offset.y);
  vec2 vGreenPos  = vec2(gl_TexCoord[0].x, gl_TexCoord[0].y);
```

```
  vec2 vBluePos   = vec2(gl_TexCoord[0].x - offset.x, gl_TexCoord[0].y -
offset.y);

  // Red pixel inside texture?
  if((vRedPos.x >= 0.0) && (vRedPos.x <= 1.0) && (vRedPos.y >= 0.0) &&
(vRedPos.y <= 1.0))
  {
    // Gets its value
    fRed = texture2D(texture, vRedPos).r;
  }

  // Green  pixel inside texture?
  if((vGreenPos.x >= 0.0) && (vGreenPos.x <= 1.0) && (vGreenPos.y >= 0.0) &&
(vGreenPos.y <= 1.0))
  {
    // Gets its value
    fGreen = texture2D(texture, vGreenPos).g;
  }

  // Blue pixel inside texture?
  if((vBluePos.x >= 0.0) && (vBluePos.x <= 1.0) && (vBluePos.y >= 0.0) &&
(vBluePos.y <= 1.0))
  {
    // Gets its value
    fBlue = texture2D(texture, vBluePos).b;
  }

  // Outputs the final decomposed pixel
  gl_FragColor = vec4(fRed, fGreen, fBlue, 1.0);
}"
ParamList = texture # offset
offset    = (-0.05, -0.05, 0.0) ~ (0.05, 0.05, 0.0); <= Let's take some
random offset
```

Please see the spawner/shader tutorial for more information.

1)
delimited by double quotes (“) as seen in the syntax page
2)
if the owner is a viewport, it will be its associated texture; if it's an object, it's current graphic/animation key's texture will be used