Shader Coordinates Tutorial

Summary

This tutorial shows how to connect shader based coordinates with the screen coordinate system.

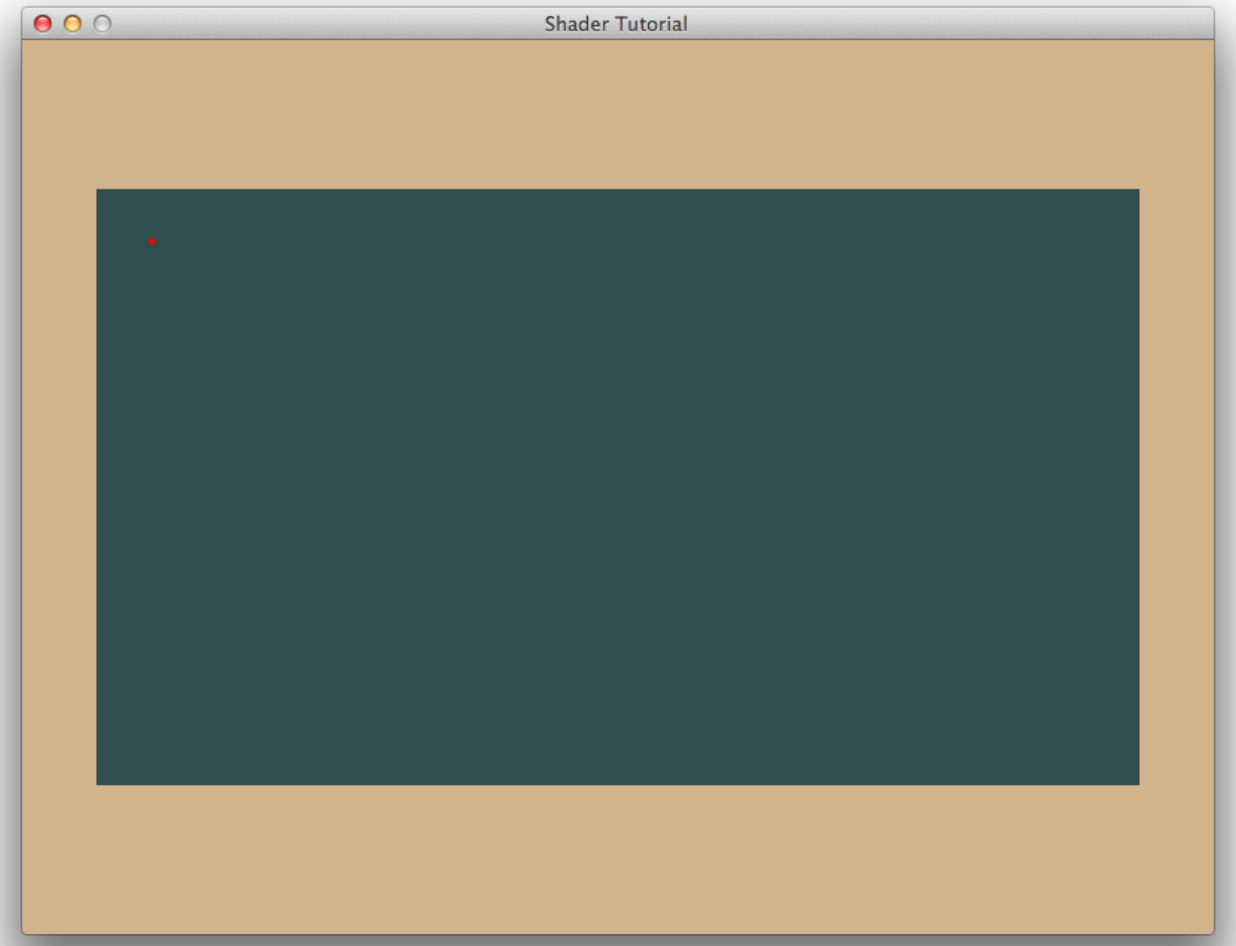
This project is intentionally made simple to allow for better focus on what it takes to work with just shaders. The technique shown here provides for the basis of handling rectangular and hexagon grids drawn in shaders.

Details

This tutorial consists of just two files. One is a standard orx .c file with main function and another file is the configuration file.

The project is written against the current version of ORX on May 7, 2013.

The following screenshot demonstrates the final application look on Mac OS X:



On the screenshot you can see a red rectangular dot. It indicates the position of the mouse cursor at the moment of taking the screenshot.

Init function

Init function contains routine initialization code. It sets up view port and loads a single object to display on the screen called Object.

The origin point of the screen is the top left corner of the window display area. The screen coordinate space is marked with tan color on the screenshot above.

The shader is responsible for drawing in the area marked with the dark green color inside the tan colorspace. The shader point of origin is bottom left corner of the green color space.

Because screen and shader coordinate systems are opposite of each other we have to translate the coordinates of the mouse. For that purpose, we collect screen height in the Init function:

```
orxFLOAT screenWidth;
orxDisplay_GetScreenSize(&screenWidth, &_screenHeight);
```

Our shader is going to take a variable and its value has to be configured by the code at run time. orx engine provides a way to inject the value into the shader configuration parameter right before the shader program is executed. Orx is event driven and shader parameter configuration is done through orx event system. Init function sets up our shader listener:

```
orxEvent_AddHandler(orxEVENT_TYPE_SHADER, handleShaderEvent);
```

Run function

Run function is part of the main application loop in orx. Run is called with every loop iteration. Handling current coordinate in the shader event handler would result in too many calls to read the mouse coordinate within a single iteration of the main loop. That's why we read input in the run function.

```
orxSTATUS orxFASTCALL Run()
```

Our example works with the screen coordinates, so a simple call the the orxMouse_GetPosition is sufficient:

```
orxVECTOR mouse;
orxMouse_GetPosition(&mouse);
_texCoord.fX = mouse.fX;
_texCoord.fY = _screenHeight - mouse.fY;
_texCoord.fZ = 0.;
```

_texCoord.fY has to be recalculated due to opposing direction of screen and shader coordinate space.

handleShaderEvent function

The shader program takes mouse coordinate as its only argument that has to be passed from the code behind.

Orx sends an event before the shader program execution for every shader program parameter. The function handleShaderEvent is registered to handle shader events (see Init function for registration).

The function “if” statement makes sure that the value is set for the appropriate parameter name and not for other or built-in parameter. Only then the parameter value is set to the mouse coordinate (_texCoord):

```
if(!orxString_Compare(pstPayload->zParamName, "mousePos")) {
    orxVector_Copy(&pstPayload->vValue, &_texCoord);
}
```

The function must report success for shader processing to take place.

Shader Configuration

Shader Program

The actual shader program is very simple. It stores currently processed pixel in a helper variable for ease of referencing. Then it compares the pixel against the current mouse position pixel coordinate. If the pixel is close to the mouse coordinate, then it is marked as red. Otherwise, the pixel is marked with the tone of the green color:

```
void main()
{
    vec3 p = gl_FragCoord.xyz;
    if (abs(mousePos.x - p.x) < 2. && abs(mousePos.y - p.y) < 2.) {
        gl_FragColor = vec4(1., 0., 0., 1.); // red
    } else {
        gl_FragColor = vec4(.184, .309, .309, 1.); // greenish
    }
}
```

Shader Configuration

Shader program itself has additional configuration parameters.

- ParamList defines names of the parameters. In this case we use mousePos parameter name.
- UseCustomParam indicates that shader parameters may be set by the code behind. Without this parameter handleShaderEvent will not get called as the event will not get triggered for this shader. [b]This parameter name is case sensitive.[/b]

- mousePos is our actual parameter. Orx uses supplied value to detect the data type associated with the parameter.

```
[Shader]
ParamList = mousePos
UseCustomParam = true
mousePos = (799., 599., 0.)
```

Code behind creates an orx object based on the configuration specified in [Object] section displayed below. This section specifies that object is represented by a one pixel graphic. The graphic is then scaled to occupy the required amount of space with the Scale parameter. In this case the scale represents width and height of the object, because underlying graphic is just 1 pixel in size.

ShaderList parameter simply references the shader section.

```
[Object]
Graphic    = OnePixel
Position   = (-350., -200., -0.)
Scale      = (700, 400, 1.)
ShaderList  = Shader
```

The word pixel to the right of the Texture parameter name in [OnePixel] section is an orx keyword that defines one pixel.

```
[OnePixel]
Texture = pixel
```

Reference Implementation of Shader.ini

```
; orx - Tutorial config file
; Should be used with orx v.1.4+

[Display]
ScreenWidth   = 800
ScreenHeight  = 600
Title         = Shader Tutorial

[Input]
SetList = MainInput

[MainInput]
KEY_ESCAPE = Quit

[Viewport]
Camera            = Camera
BackgroundColor   = (210, 180, 140)
```

```
[Camera]
; We use the same size for the camera than our display on screen so as to
obtain a 1:1 ratio
FrustumWidth  = @Display.ScreenWidth
FrustumHeight = @Display.ScreenHeight
FrustumFar    = 1.0
FrustumNear   = 0.0
Position      = (0.0, 0.0, -1.0)

[Object]
Graphic    = OnePixel
Position   = (-350., -200., -0.)
Scale      = (700, 400, 1.)
ShaderList  = Shader

[OnePixel]
Texture = pixel

[Shader]
ParamList = mousePos
UseCustomParam = true
mousePos = (799., 599., 0.)
Code = "
void main()
{
    vec3 p = gl_FragCoord.xyz;
    if (abs(mousePos.x - p.x) < 2. && abs(mousePos.y - p.y) < 2.) {
        gl_FragColor = vec4(1., 0., 0., 1.);
    } else {
        gl_FragColor = vec4(.184, .309, .309, 1.);
    }
}
"
```

Reference Implementation of Shader.c

```
/* Orx - Portable Game Engine
 *
 * Copyright (c) 2008-2010 Orx-Project
 *
 * This software is provided 'as-is', without any express or implied
 * warranty. In no event will the authors be held liable for any damages
 * arising from the use of this software.
 *
 * Permission is granted to anyone to use this software for any purpose,
 * including commercial applications, and to alter it and redistribute it
 * freely, subject to the following restrictions:
 *
```

```
 *    1. The origin of this software must not be misrepresented; you must not
 *    claim that you wrote the original software. If you use this software
 *    in a product, an acknowledgment in the product documentation would be
 *    appreciated but is not required.
 *
 *    2. Altered source versions must be plainly marked as such, and must not be
 *    misrepresented as being the original software.
 *
 *    3. This notice may not be removed or altered from any source
 *    distribution.
 */

/**
 * @file 13_Shader.c
 * @date 05/07/2013
 * @author Sergei G
 *
 * Mouse tracking and shader.
 */


#include "orx.h"


/* This is a basic C tutorial creating a viewport and an object.
 *
 * As orx is data driven, here we just write 2 lines of code to create a viewport
 * and an object. All their properties are defined in the config file (01_Object.ini).
 * As a matter of fact, the viewport is associated with a camera implicitly created from the
 * info given in the config file. You can also set their sizes, positions, the object colors,
 * scales, rotations, animations, physical properties, and so on. You can even request
 * random values for these without having to add a single line of code.
 * In a later tutorial we'll see how to generate your whole scene (all background
 * and landscape objects for example) with a simple for loop written in 3 lines of code.
 *
 * For now, you can try to uncomment some of the lines of 01_Object.ini, play with them,
 * then relaunch this tutorial. For an exhaustive list of options, please look at CreationTemplate.ini.
```

```
 */

orxFLOAT _screenHeight;
orxVECTOR _texCoord;

static orxSTATUS orxFASTCALL handleShaderEvent(const orxEVENT *currentEvent)
{
    switch(currentEvent->eID){
        case orxSHADER_EVENT_SET_PARAM: {
            /* Gets its payload */
            orxSHADER_EVENT_PAYLOAD *pstPayload = (orxSHADER_EVENT_PAYLOAD
*)currentEvent->pstPayload;
            /* look for parameter of interest */
            if(!orxString_Compare(pstPayload->zParamName, "mousePos")) {
                orxVector_Copy(&pstPayload->vValue, &_texCoord);
            }
        }
    }
    return orxSTATUS_SUCCESS;
}

/** Inits the tutorial
 */
orxSTATUS orxFASTCALL Init()
{
    /* Displays a small hint in console */
    orxLOG("\n* This tutorial creates a viewport/camera couple and an object
with shader"
           "\n* You can play with the config parameters in ../13_Shader.ini"
           "\n* After changing them, relaunch the tutorial to see their
effects");

    orxFLOAT screenWidth;
    orxDisplay_GetScreenSize(&screenWidth, &_screenHeight);

    orxViewport_CreateFromConfig("Viewport");
    orxObject_CreateFromConfig("Object");
    orxEvent_AddHandler(orxEVENT_TYPE_SHADER, handleShaderEvent);
    return orxSTATUS_SUCCESS;
}

/** Run function
 */
orxSTATUS orxFASTCALL Run()
{
    orxSTATUS eResult = orxSTATUS_SUCCESS;

    /* Should quit? */
    if(orxInput_IsActive("Quit"))
    {
        /* Updates result */
```

```
        eResult = orxSTATUS_FAILURE;
    }
    /* stores current mouse position */
    orxVECTOR mouse;
    orxMouse_GetPosition(&mouse);
    _texCoord.fX = mouse.fX;
    _texCoord.fY = _screenHeight - mouse.fY;
    _texCoord.fZ = 0.;
    /* Done! */
    return eResult;
}

/** Exit function
 */
void orxFASTCALL Exit()
{
    /* We're a bit lazy here so we let orx clean all our mess! :) */
}

/** Main function
 */
int main(int argc, char **argv)
{
    /* Executes a new instance of tutorial */
    orx_Execute(argc, argv, Init, Run, Exit);

    return EXIT_SUCCESS;
}
```