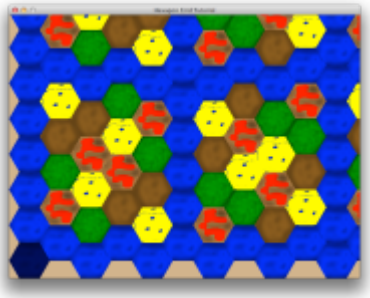
Hexagon Grid Tutorial

Introduction

This tutorial demonstrates how to generate hex grid using shaders and how to track hexagon tiles based on the screen coordinate (mouse cursor tracking).



The darker hexagon in the left bottom corner of the screenshot above marks the mouse position when screenshot was taken.

This tutorial builds on concepts developed in Shader Coordinates Tutorial. The enhancement in this tutorial is about more complicated hexagon grid shader and more complicated internal world coordinate system in code behind.

NOTE: An updated version of this tutorial, that is based on axial/cubial coordinates, can be found at: Hexagon Grid Tutorial (Axial/Cubial Coordinates)

Details

Hexagon grid math is based on the article in http://blog.ruslans.com/2011/02/hexagonal-grid-math.html blog post.

TBD: Tile resource files.

Loading parameter with Orx Config API

The value of hexagon radius is defined in INI file. Code behind needs radius for internal calculations thus it has to load the value from the INI file. It is done with the help of orxConfig_XXXX module functions:

```
orxConfig_PushSection("Shader");
_radius = orxConfig_GetFloat("radius");
orxConfig_PopSection();
```

The push section call loads or selects the INI file section marked with the name [Shader].

```
orxConfig_PushSection("Shader");
```

Once the section is selected values inside the section can be accessed with a family of function calls orxConfig_GetXXXX. In our case we simply load a signed integer value and store it in module scope variable:

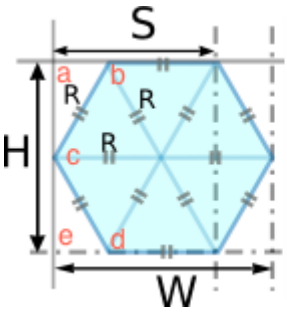
```
_screenHeight = orxConfig_GetS32("radius");
```

Configuration state is restored to its original value with call to pop section:

```
orxConfig_PopSection();
```

Visualizing Hexagon Parameters

The code uses variable names based on the image below:



$$W = 2 \cdot R$$
$$S = \frac{3}{2} \cdot R$$
$$H = 2 \cdot R \cdot sin(60°) = \sqrt{3} \cdot R$$

Source Code

The structure of the source code is similar to the structure of tutorial projects that come with Orx source code.

INI File HexagonGrid.ini

The content of the file is split into 3 blocks for syntax highlighting reason.

Block 1 - Most of INI content:

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

[Display]
ScreenWidth    = 800
ScreenHeight   = 600
Title          = Hexagon Grid 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   = (-400., -300., -0.)
Scale      = (800, 600, 1.)
ShaderList  = Shader

[OnePixel]
Texture = pixel

[Shader]
ParamList = radius # textures # texturesCount # highlight
UseCustomParam = true
textures = ../data/hexagonTiles/tile-blue.png # ../data/hexagonTiles/tile-
red.png # ../data/hexagonTiles/tile-green.png # ../data/hexagonTiles/tile-
brown.png # ../data/hexagonTiles/tile-yellow.png
texturesCount = 5
radius = 46.66
highlight = (0., 1., 0.)
Code = "
```

Block 2 - Shader code is in its own code block for syntax highlighting:

```
bool sameSide(vec3 p1, vec3 p2, vec3 a, vec3 b) {
    vec3 cp1 = cross(b-a, p1-a);
    vec3 cp2 = cross(b-a, p2-a);
    return dot(cp1, cp2) >= 0.0;
}

bool pointInTriangle(vec3 p, vec3 a, vec3 b, vec3 c) {
    return sameSide(p,a, b,c)
        && sameSide(p,b, a,c)
        && sameSide(p,c, a,b);
}
```

```
vec4 tileColor(vec2 tile, float s, float h) {
    if (tile.x < 0.0 || tile.y < 0.0)
        return vec4(0.0, 0.0, 0.0, 1.0);
    return vec4(tile.x * s, tile.y * h, .7, 1.0); // red
}

// This one works by producing a nice hexagon greed.
void main()
{
    float PI = 3.14159265358979323846264;
    float r = radius;
    float h = 2.0 * r * sin(PI / 3.0); // PI / 3.0 is a 60 degree angle
    float s = 3. / 2. * r;
    vec3 p = gl_FragCoord.xyz; // current point

    // aproximation or dirty grid coordinates
    float gridX = floor(p.x / s);
    float yOffset = mod(gridX, 2.0)*0.5*h;
    float gridY = floor((p.y - yOffset) / h);
    vec3 a = vec3(gridX * s, gridY * h + yOffset, 0.0); // top left corner
of the top outside triangle
    vec3 b = vec3(a.x + 0.5 * r, a.y, 0.0); // top right cornerof the top
outside triangle
    vec3 c = vec3(a.x, a.y + 0.5 * h, 0.0); // bottom point of the top
outside triangle
    vec2 tile;
    vec2 origin; // tile origin point
    if (pointInTriangle(p, a,b,c)) {
        // outside top left corner
        tile = vec2(gridX - 1., gridY - mod(gridX+1., 2.));
        origin = vec2(a.x - s, a.y + 0.5 * h);
    } else {
        vec3 d = vec3(b.x, (gridY + 1.) * h + yOffset, 0.0);
        vec3 e = vec3(a.x, d.y, 0.0);
        if (pointInTriangle(p, c,d,e)) {
            // outside bottom left corner
            tile = vec2(gridX - 1., gridY + mod(gridX, 2.));
            origin = vec2(c.x - s, c.y + h);
        } else {
            // main part or current tile
            tile = vec2(gridX, gridY);
            origin = e.xy;
        }
    }

    vec2 textCoord = vec2(mod(p.x-origin.x, 2.0*r) / 2. / r, mod(p.y-
origin.y, h) / h);

    int idx = int(mod(tile.x * tile.y, texturesCount));
```

```
    vec4 color = texture2D(textures[idx], textCoord);
    if (highlight.xy == tile) {
        // shade the pixel
        color = mix(color, vec4(.0, .0, .1, 1.), .7);
    }
    if (tile.x == -1. || tile.y == -1.)
        discard;
    else
        gl_FragColor = color;
}
```

Block 3 contains the end of INI file:

```
"
```

C File HexagonGrid.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;
orxFLOAT _radius; // tile radius in screen coordinates, i.e. pixels
orxVECTOR _tilePos; // tile position index

#pragma mark - hexagon math

orxBOOL sameSide(orxVECTOR *p1, orxVECTOR *p2, orxVECTOR *a, orxVECTOR *b) {
    orxVECTOR b_a, p1_a, p2_a, cp1, cp2;
    orxVector_Sub(&b_a, b, a);
    orxVector_Sub(&p1_a, p1, a);
    orxVector_Sub(&p2_a, p2, a);
    orxVector_Cross(&cp1, &b_a, &p1_a);
    orxVector_Cross(&cp2, &b_a, &p2_a);
    return orxVector_Dot(&cp1, &cp2) >= 0.0;
}

orxBOOL pointInTriangle(orxVECTOR *p, orxVECTOR *a, orxVECTOR *b, orxVECTOR *c) {
    return sameSide(p,a, b,c) && sameSide(p,b, a,c) && sameSide(p,c, a,b);
```

```
}

orxFLOAT mod(orxFLOAT x, orxFLOAT y) {
    return x - y * floor(x/y);
}

// This one works by producing a nice hexagon greed.
// r is radius
// p is current pixel coordinate to process
orxVECTOR* tileFromScreen(orxFLOAT r, orxVECTOR *point, orxVECTOR *tile)
{
    orxVECTOR p;
    orxVector_Copy(&p, point);

    orxFLOAT PI = 3.14159265358979323846264;
    orxFLOAT h = 2.0 * r * sin(PI / 3.0); // PI / 3.0 is a 60 degree angle
    orxFLOAT s = 3. / 2. * r;

    // aproximation or dirty grid coordinates
    orxFLOAT gridX = floor(p.fX / s);
    orxFLOAT yOffset = mod(gridX, 2.0)*0.5*h;
    orxFLOAT gridY = floor((p.fY - yOffset) / h);
    orxVECTOR a, b, c;
    orxVector_Set(&a, gridX * s, gridY * h + yOffset, 0.0); // top left
corner of the top outside triangle
    orxVector_Set(&b, a.fX + 0.5 * r, a.fY, 0.0); // top right cornerof the
top outside triangle
    orxVector_Set(&c, a.fX, a.fY + 0.5 * h, 0.0); // bottom point of the top
outside triangle

    if (pointInTriangle(&p, &a,&b,&c)) {
        // outside top left corner
        orxVector_Set(tile, gridX - 1., gridY - mod(gridX+1., 2.), 0.0);
    } else {
        orxVECTOR d, e;
        orxVector_Set(&d, b.fX, (gridY + 1.) * h + yOffset, 0.0);
        orxVector_Set(&e, a.fX, d.fY, 0.0);
        if (pointInTriangle(&p, &c,&d,&e)) {
            // outside bottom left corner
            orxVector_Set(tile, gridX - 1., gridY + mod(gridX, 2.), 0.0);
        } else {
            // main part or current tile
            orxVector_Set(tile, gridX, gridY, 0.0);
        }
    }
    return tile;
}

#pragma mark - orx

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, "highlight")) {
                orxVector_Copy(&pstPayload->vValue, &_tilePos);
            }
        }
    }
    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
../14_HexagonGrid.ini"
           "\n* After changing them, relaunch the tutorial to see their
effects");

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

    orxConfig_PushSection("Shader");
    _radius = orxConfig_GetFloat("radius");
    orxConfig_PopSection();

    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.;
    // calculate tile position for the mouse position
    orxVECTOR oldTilePos;
    orxVector_Copy(&oldTilePos, &_tilePos);
    tileFromScreen(_radius, &_texCoord, &_tilePos);
    if (oldTilePos.fX != _tilePos.fX || oldTilePos.fY != _tilePos.fY) {
        //orxLOG("tile: %f, %f for shader: %f, %f.", _tilePos.fX,
_tilePos.fY, _texCoord.fX, _texCoord.fY);
    }
    /* 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;
}
```

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/en/tutorials/shaders/hexagongrid?rev=1598889115

Last update: **2025/09/30 17:26 (14 months ago)**