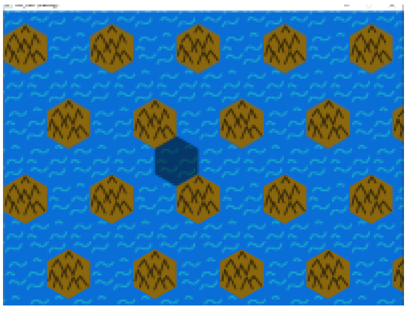
Hexagon Grid Tutorial (Axial/Cubial Coordinates)

Introduction

This tutorial showcases how to draw a hexagon grid using a shader, as well as using the mouse position to highlight specific hexagon tiles.



It is based off of the this Hexagon Grid Tutorial, the key difference being that this example makes use of an axial/cubial coordinate system for pixel-to-hex calculations, as opposed to the grid-based system used by the old tutorial.

Details

The axial/cubial coordinate system and associated mathematics used in this example are derived from the theory presented in this blog post: https://www.redblobgames.com/grids/hexagons/

Source Code

A complete version of this tutorial's source code can be found at the following git repository: https://github.com/LudiG/tut_hex

This repository also contains hexagon resource files that can be used as textures for your hexagon shader.

NOTE: Pointy-top and flat-top hexagons use different texture files, so you will need to ensure that you match the right files with your hexagon layout.

INI File (tut_hex.ini)

ORX Config File

```
[Display]
```

```
ScreenWidth   = 800
ScreenHeight  = 600
Title         = Hexagon Grid Tutorial

[Input]
SetList = MainInput

[MainInput]
KEY_ESCAPE = Quit

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

[Camera]
FrustumWidth  = @Display.ScreenWidth
FrustumHeight = @Display.ScreenHeight
FrustumFar    = 1.0
FrustumNear  = 0.0
Position      = (0.0, 0.0, -1.0)

[Object]
Graphic    = OnePixel
Position   = (-400.0, -300.0, 0.0)
Scale      = (800, 600, 1.0)
ShaderList  = Shader

[OnePixel]
Texture = pixel

[Resource]
Texture = ../data/texture

[Shader]
ParamList = radius # highlight # textures # texturesCount
UseCustomParam = true
radius = 50.0
highlight = (0.0, 0.0, 0.0)
textures = Hex_Water_PointyTop.png # Hex_Earth_PointyTop.png
texturesCount = 2
Code = "
```

GLSL Shader Code

```
#define HEX_SIZE radius

#define HEX_WIDTH_POINTYTOP (sqrt(3.0) * HEX_SIZE)
#define HEX_WIDTH_FLATTOP (2 * HEX_SIZE)

#define HEX_HEIGHT_POINTYTOP (2 * HEX_SIZE)
```

```
#define HEX_HEIGHT_FLATTOP (sqrt(3.0) * HEX_SIZE)

// Function to convert cubial coords to axial coords.

vec2 cubeToAxial(vec3 cube)
{
    return vec2(cube.x, cube.y);
}

// Function to convert axial coords to cubial coords.

vec3 axialToCube(vec2 axial)
{
    float x = axial.x;
    float y = axial.y;

    float z = -x - y;

    return vec3(x, y, z);
}

// Function to round float cubial coords to int cubial coords.

vec3 cubeRound(vec3 cube)
{
    int rx = int(round(cube.x));
    int ry = int(round(cube.y));
    int rz = int(round(cube.z));

    float xDiff = abs(rx - cube.x);
    float yDiff = abs(ry - cube.y);
    float zDiff = abs(rz - cube.z);

    if ((xDiff > yDiff) && (xDiff > zDiff))
        rx = -ry - rz;

    else if (yDiff > zDiff)
        ry = -rx - rz;

    else
        rz = -rx - ry;

    return vec3(rx, ry, rz);
}

// Function to round float axial coords to int axial coords.

vec2 axialRound(vec2 axial)
{
    return cubeToAxial(cubeRound(axialToCube(axial)));
}
```

```
// Function to return axial hex-grid coords, given a screen position (horizontal, pointy-top hex layout).

vec2 pixelToHex_PointyTop(vec2 point)
{
    return vec2(((sqrt(3.0)/3.0 * point.x) + (-1.0/3.0 * point.y)) / HEX_SIZE, (2.0/3.0 * point.y) / HEX_SIZE);
}

// Function to return axial hex-grid coords, given a screen position (vertical, flat-top hex layout).

vec2 pixelToHex_FlatTop(vec2 point)
{
    return vec2((2.0/3.0 * point.x) / HEX_SIZE, ((-1.0/3.0 * point.x) + (sqrt(3.0)/3.0 * point.y)) / HEX_SIZE);
}

// Function to return a screen position, given axial hex-grid coords (horizontal, pointy-top hex layout).

vec2 hexToPixel_PointyTop(vec2 hex)
{
    return vec2(((sqrt(3.0) * hex.x) + (sqrt(3.0)/2.0 * hex.y)) * HEX_SIZE, (3.0/2.0 * hex.y) * HEX_SIZE);
}

// Function to return a screen position, given axial hex-grid coords (vertical, flat-top hex layout).

vec2 hexToPixel_FlatTop(vec2 hex)
{
    return vec2((3.0/2.0 * hex.x) * HEX_SIZE, ((sqrt(3.0)/2.0 * hex.x) + (sqrt(3.0) * hex.y)) * HEX_SIZE);
}

// Main shader.

void main()
{
    vec2 point = vec2(gl_FragCoord.x, gl_FragCoord.y);

    vec2 hex = axialRound(pixelToHex_PointyTop(point));

    float width = HEX_WIDTH_POINTYTOP;
    float height = HEX_HEIGHT_POINTYTOP;

    vec2 center = hexToPixel_PointyTop(hex);
    vec2 origin = vec2(center.x - (width/2.0), center.y - (height/2.0));
    vec2 textureCoord = vec2(((point.x - origin.x) / width), 1.0 - ((point.y
- origin.y) / height));

    int index = int(mod(hex.x * hex.y, texturesCount));
    vec4 color = texture2D(textures[index], textureCoord);

    if (highlight.xy == hex)
        color = mix(color, vec4(0.0, 0.0, 0.0, 1.0), 0.5);

    gl_FragColor = color;
}
```

```
"
```

CPP File (tut_hex.cpp)

```
/**
 * @file tut_hex.cpp
 * @date 2019/09/06
 * @author LudiG
 *
 * Axial/Cubial coord-based hexagon shader and mouse tracker.
 */

/* 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.
 */
```

```
#include "orx.h"

orxFLOAT _screenHeight; // The screen height.
orxFLOAT _screenWidth; // The screen width.

orxFLOAT _tileRadius; // The tile radius in screen coordinates (pixels).

orxVECTOR _screenCoord; // The current screen coordinates of the mouse.
orxVECTOR _tilePos; // The current tile position.

// HEX

// Function to convert cubial coords to axial coords.

orxVECTOR cubeToAxial(const orxVECTOR& cube)
{
    orxVECTOR result;
    orxVector_Set(&result, cube.fX, cube.fY, 0.0);

    return result;
}

// Function to convert axial coords to cubial coords.

orxVECTOR axialToCube(const orxVECTOR& axial)
{
    orxFLOAT x = axial.fX;
    orxFLOAT y = axial.fY;

    orxFLOAT z = -x - y;

    orxVECTOR result;
    orxVector_Set(&result, x, y, z);

    return result;
}

// Function to round float cubial coords to int cubial coords.

orxVECTOR cubeRound(const orxVECTOR& cube)
{
    orxFLOAT rx = orxMath_Round(cube.fX);
    orxFLOAT ry = orxMath_Round(cube.fY);
    orxFLOAT rz = orxMath_Round(cube.fZ);

    orxFLOAT xDiff = orxMath_Abs(rx - cube.fX);
    orxFLOAT yDiff = orxMath_Abs(ry - cube.fY);
    orxFLOAT zDiff = orxMath_Abs(rz - cube.fZ);
    if ((xDiff > yDiff) && (xDiff > zDiff))
        rx = -ry - rz;

    else if (yDiff > zDiff)
        ry = -rx - rz;

    else
        rz = -rx - ry;

    orxVECTOR result;
    orxVector_Set(&result, rx, ry, rz);

    return result;
}

// Function to round float axial coords to int axial coords.

orxVECTOR axialRound(const orxVECTOR& axial)
{
    return cubeToAxial(cubeRound(axialToCube(axial)));
}

// Function to return axial hex-grid coords, given a screen position (horizontal, pointy-top hex layout).

orxVECTOR pixelToHex_PointyTop(const orxVECTOR& point)
{
    orxFLOAT size = _tileRadius;

    orxVECTOR result;
    orxVector_Set(&result, ((orxMath_Pow(3.0, 0.5)/3.0 * point.fX) + (-1.0/3.0 * point.fY)) / size, (2.0/3.0 * point.fY) / size, 0.0);

    return result;
}

// Function to return axial hex-grid coords, given a screen position (vertical, flat-top hex layout).

orxVECTOR pixelToHex_FlatTop(const orxVECTOR& point)
{
    orxFLOAT size = _tileRadius;

    orxVECTOR result;
    orxVector_Set(&result, (2.0/3.0 * point.fX) / size, ((-1.0/3.0 * point.fX) + (orxMath_Pow(3.0, 0.5)/3.0 * point.fY)) / size, 0.0);

    return result;
}

// Function to return a screen position, given axial hex-grid coords
```

```
(horizontal, pointy-top hex layout).

orxVECTOR hexToPixel_PointyTop(const orxVECTOR& hex)
{
    orxFLOAT size = _tileRadius;

    orxVECTOR result;
    orxVector_Set(&result, ((orxMath_Pow(3.0, 0.5)  * hex.fX) + (orxMath_Pow(3.0, 0.5)/2.0 * hex.fY)) * size, (3.0/2.0 * hex.fY) * size, 0.0);

    return result;
}

// Function to return a screen position, given axial hex-grid coords (vertical, flat-top hex layout).

orxVECTOR hexToPixel_FlatTop(const orxVECTOR& hex)
{
    orxFLOAT size = _tileRadius;

    orxVECTOR result;
    orxVector_Set(&result, (3.0/2.0 * hex.fX) * size, ((orxMath_Pow(3.0, 0.5)/2.0 * hex.fX) + (orxMath_Pow(3.0, 0.5)  * hex.fY)) * size, 0.0);

    return result;
}

// ORX

static orxSTATUS orxFASTCALL handleShaderEvent(const orxEVENT* currentEvent)
{
    switch(currentEvent->eID)
    {
        case orxSHADER_EVENT_SET_PARAM:
        {
            // Get the event payload.
            orxSHADER_EVENT_PAYLOAD *pstPayload = (orxSHADER_EVENT_PAYLOAD*)currentEvent->pstPayload;

            // look for the parameter of interest.
            if (!orxString_Compare(pstPayload->zParamName, "highlight"))
                orxVector_Copy(&pstPayload->vValue, &_tilePos);
        }
    }

    return orxSTATUS_SUCCESS;
}

orxSTATUS orxFASTCALL Init()
```

```
{
    orxDisplay_GetScreenSize(&_screenWidth, &_screenHeight);

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

    orxViewport_CreateFromConfig("Viewport");
    orxObject_CreateFromConfig("Object");

    orxEvent_AddHandler(orxEVENT_TYPE_SHADER, handleShaderEvent);

    return orxSTATUS_SUCCESS;
}

orxSTATUS orxFASTCALL Run()
{
    orxSTATUS result = orxSTATUS_SUCCESS;

    // INPUT: Quit
    if(orxInput_IsActive("Quit"))
        result = orxSTATUS_FAILURE;

    // INPUT: Mouse
    orxVECTOR mouse;
    orxMouse_GetPosition(&mouse);

    _screenCoord.fX = mouse.fX;
    _screenCoord.fY = _screenHeight - mouse.fY;
    _screenCoord.fZ = 0.0;

    // Calculate the tile position from the mouse position.
    orxVECTOR tilePosOld;
    orxVector_Copy(&tilePosOld, &_tilePos);
    _tilePos = axialRound(pixelToHex_PointyTop(_screenCoord));

    // Print tile and screen position if mouse moves.
    if ((tilePosOld.fX != _tilePos.fX) || (tilePosOld.fY != _tilePos.fY))
        orxLOG("TILE: %f, %f FOR SHADER: %f, %f.", _tilePos.fX, _tilePos.fY,
_screenCoord.fX, _screenCoord.fY);

    return result;
}

void orxFASTCALL Exit()
{
    // No specific garbage-collection requirements.
}

orxSTATUS orxFASTCALL Bootstrap()
{
    // Add the config directory as a resource path.
    orxResource_AddStorage(orxCONFIG_KZ_RESOURCE_GROUP, "../data/config",
orxFALSE);

    return orxSTATUS_SUCCESS;
}

int main(int argc, char** argv)
{
    orxConfig_SetBootstrap(Bootstrap);

    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/hexagongrid2?rev=1598883083

Last update: **2025/09/30 17:26 (7 months ago)**