Maps in a Shader

TiledToOrx, OrxImageMap and Tilemaps in a shader, what's it all about?

Long scrolling levels can be constructed using graphic tiles laid out in a large map. Normally, you would use a paint routine to read all your tilemap information from a data configuration file, and render them as objects to your scene.

Here's an example of a very simple tile map defined in a data configuration file:

We could have a simple TileSet image with four coloured 32×32 blocks:



And the data config to use it in a small map would look something like this.

```
[Block]
Graphic       = @
Texture       = block.png
TextureSize   = (32, 32)

[Block1@Block]
TextureOrigin = (0, 0)

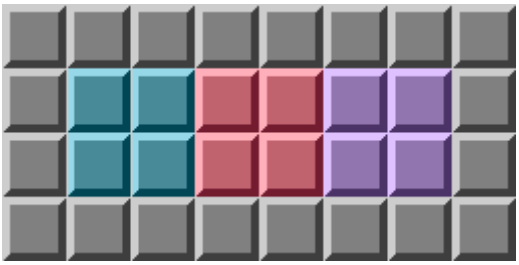
[Block2@Block]
TextureOrigin = (32, 0)

[Block3@Block]
TextureOrigin = (0, 32)

[Block4@Block]
TextureOrigin = (32, 32)

[GameMap]
MapRow1       = Block1 # Block1 # Block1 # Block1 # Block1 # Block1 # Block1
# Block1
MapRow2       = Block1 # Block2 # Block2 # Block3 # Block3 # Block4 # Block4
# Block1
MapRow3       = Block1 # Block2 # Block2 # Block3 # Block3 # Block4 # Block4
# Block1
MapRow4       = Block1 # Block1 # Block1 # Block1 # Block1 # Block1 # Block1
# Block1
```

This is a contrived example. We could write a small routine to display the above map to the screen:



This will work fine in most games, but there will be a point where thousands of objects will start to weight the CPU down. Even simple objects still have to be tracked by Orx.

Using a shader and a neat trick to render the entire map at once using the GPU, you can have extremely long and smooth scrolling maps, all done by the GPU and with very little overhead. This idea was demonstrated in the Tilemap demo application at: https://github.com/iarwain/tilemap. In fact, this article and the tooling is all based on that project.

How does it work?

The core of the idea is simple: instead of having data configuration for the map, convert the map data to a special bitmap instead. You can't pass thousands of lines of data configuration to a shader, but you can supply an image texture.

The shader can then take this bitmap with tileset index values encoded into it, and translate these values into tile indexes and then compute the tile position inside the tileset image. Yes there are two images: one to hold all the tiles (the Tileset) and our special texture which contains the “map” index data.

The shader (which is supplied with the converter application coming up shortly) will paint the tiles every frame, and scroll them for you.

The workflow

1. Create or locate a Tileset image. You can find many free ones to try at OpenGameArt
2. Create a Map using Tiled.
3. Export the tiled map to a TMX file.
4. Create a new Orx project.
5. Use TiledToOrx (at least version 0.71) to convert the TMX into Orx configuration data.
6. Create an empty data config file in your projects data eg. `tiled.ini` to hold the map data temporarily.
7. Copy the config data from TiledToOrx, and paste into the `tiled.ini` in your project.
8. Use the OrxImageMap https://gitlab.com/sausagejohnson/orximagemap program to create an image map file and a `shader.ini` file for use in your project. OrxImageMap is a command line project. Example usage:

orximagemap -i C:\Dev\orx-projects\MyScrollingGame\data\config\tiled.ini -m “GameMap” -o C:\Dev\orx-projects\MyScrollingGame\data\texture\map-image.png -s C:\Dev\orx-

projects\MyScrollingGame\data\config\shader.ini

```
The -i argument is the input tiled.ini saved to your project. This is where
the map information is sourced.
The -m is the Map section name in the data config. The converter starts
enumerating the map from there.
The -o argument is where to save the map image that the shader will use.
The -s argument is where to save the sample shader code that you will need
for your game.

- Add @shader.ini@ to the bottom of your project's main data config file to
ensure the shader.ini is included.
- Add the following to your main data config file:
```

```
        [Scene]
        ChildList       = Map

        [Map]
        Graphic         = GameMap
        ShaderList      = GameMap
        ParentCamera    = MainCamera
        UseParentSpace  = both
        Scale           = 1
        Position        = (0, 0, 1)
```

Set CameraPos to wherever it needs to start.

Compile and run. Your map should display inside the shader and the scrolling should be super smooth no matter how big the map is.

All should work. You can optionally remove the map data rows that are included in the shader section. They're not needed by the shader. The image map passed to the shader replaces the need for this now.

That being said, you could keep the map data there in case you need to still create any physical objects on the map the traditional way.

Workflow Tips

If you are going to make frequent changes between Tiled and your Orx project, remember that the the TiledToOrx project will automatically display the latest converted data config. So you could easily copy/paste the new data into your tiled.ini cofig and have a a batch file that executes the orximagemap program to build the latest image map file. But don't re-export the shader file or it will overwrite any changes you might make. A suggested ongoing command line use for a batch file would be:

```
  orximagemap -i C:\Dev\orx-projects\MyScrollingGame\data\config\tiled.ini -
m "GameMap" -o C:\Dev\orx-projects\MyScrollingGame\data\texture\map-
image.png
```

Further reading

https://github.com/iarwain/tilemap

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

Permanent link:
https://www.orx-project.org/wiki/en/tutorials/shaders/maps_in_a_shader?rev=1604220155

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