Revolution

2D Graphics Library (G2D)

Version 1.00

The contents in this document are highly confidential and should be handled accordingly.

RVL-06-0036-001-B
Released: October 22, 2007

Contents

Code Examples

Figures

Revision History

Version	Date Revised	Item	Description
1.00	2006/03/01	-	First release by Nintendo of America Inc.

1 Overview

We provide the Revolution 2D Graphics API (G2D) in source code form. The main focus of the API is tile-based 2D games. The API includes the following features:

- Supports multiple independently-scrollable layers of tiles
- Supports layers of different sizes (that is, layer width and height values can be any power of 2 up to the limitations of RAM)
- Allows a different tile size for each layer (i.e., tile width and height can be 16, 32, 64, 128, 256, 512, or 1024)
- Layers may be wrapped or non-wrapped independently
- Tiles within a single layer can be chosen from multiple source textures (but they must be of a fixed size)
- Supports all normal texture formats, as well as tiles using only color or empty tiles which are not rendered
- Supports rotation with “free” antialiasing, provided that tiles are designed with a border color restriction (see "3 Graphics Requirements" on page 9)
- Supports alpha blending
- Supports two formats for map data—either 1 byte per index or 2 bytes per index
- Allows viewport to be specified
- Optimizes display performance by determining the visible tiles intersecting the viewport, sorting by tile type and rendering with a minimal amount of state-flipping in the graphics pipeline

"Figure 1–1 G2D API Overview" on page 6 shows an overview of the relationships in the API. The diagram is a little complicated but it summarizes the entire structure of a layer.

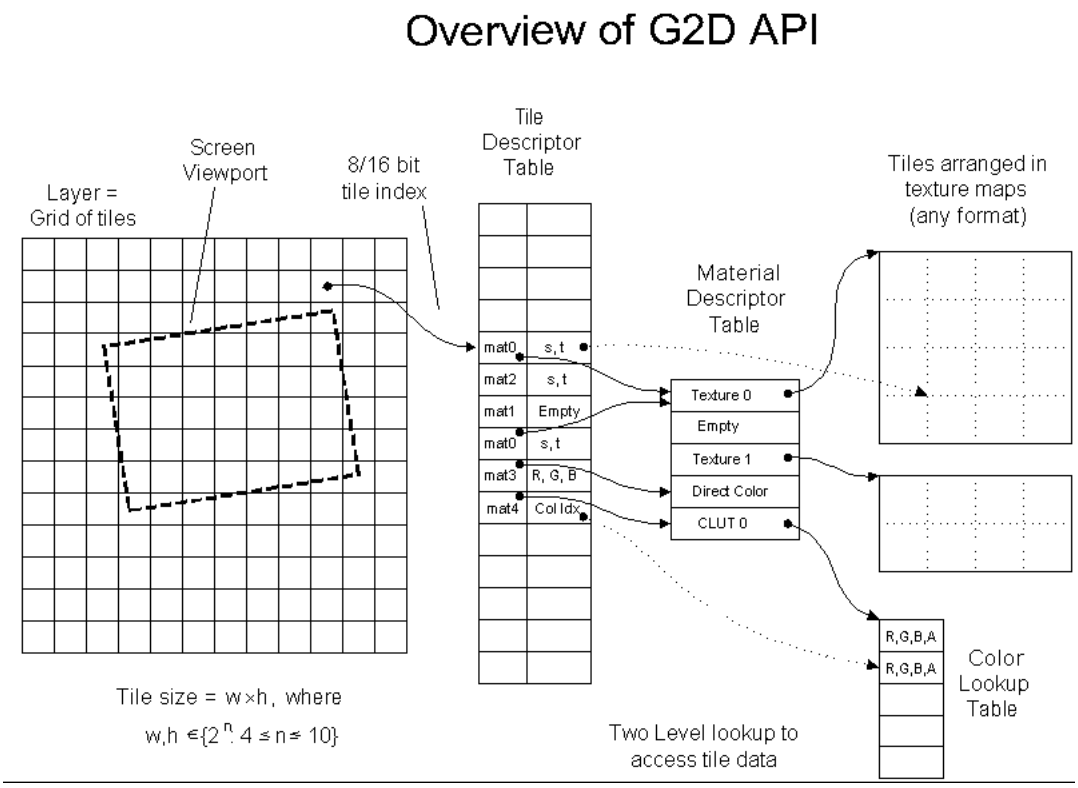
A layer is composed of a grid of tiles (that is, an array of tile indices). A layer can use either 8-bit indices or 16-bit indices depending on the total number of distinct tiles used in the layer.

The tile index references a Tile Descriptor Table. This table stores a material index for each tile, and other data associated with the tile.

The material index references a Material Descriptor Table, which stores the category of the material and the address of the source data (texture map or color lookup table) for tiles using that material. When rendering, the API sorts tiles by material index so that all tiles with the same material index located within the viewport are rendered together. Textured materials can have an optional material color, in which case the material color is modulated by the texture data to produce the final output color at each pixel.

For efficiency, all width and height values in this API are restricted to powers of 2. This includes tile sizes, texture map sizes, layer sizes and world space size. This improves texture cache coherency by ensuring that layers will wrap properly with respect to world coordinates.

Figure 1–1 G2D API Overview



2 Functions

The G2D API includes the following functions:

Code 2–1 G2D API

```
void G2DInitWorld( u32 nWorldX, u32 nWorldY );
void G2DSetViewport( u16 nTlcX, u16 nTlcY, u16 nWidth, u16 nHeight );
void G2DSetCamera( G2DPosOri *po );
void G2DDrawLayer( G2DLayer *layer, s8 *aSortBuffer );

void G2DInitSprite( G2DSprite *sprite );
void G2DDrawSprite( G2DSprite *sprite, G2DPosOri *posOri );
```

`G2DInitWorld` specifies the dimensions of the 2D world space. *nWorldX* and *nWorldY* are measured in pixels and must be a power of 2.

`G2DSetViewport` sets the viewing rectangle for rendering. *nLeft* is the left *x*-coordinate of the viewport in screen space, measured in pixels. *nTop* is the top *y*-coordinate of the viewport in screen space, measured in pixels. *nWidth* is the width of the viewport, and *nHeight* is the height of the viewport, both measured in pixels.

`G2DSetCamera` sets the current position and orientation of the camera. The `G2DposOri` structure passes the position and orientation. The *rOriX* and *rOriY* components of this structure specify the orientation as an Up vector. This vector is the direction in world space that will be rendered as upwards in screen space. It must be normalized.

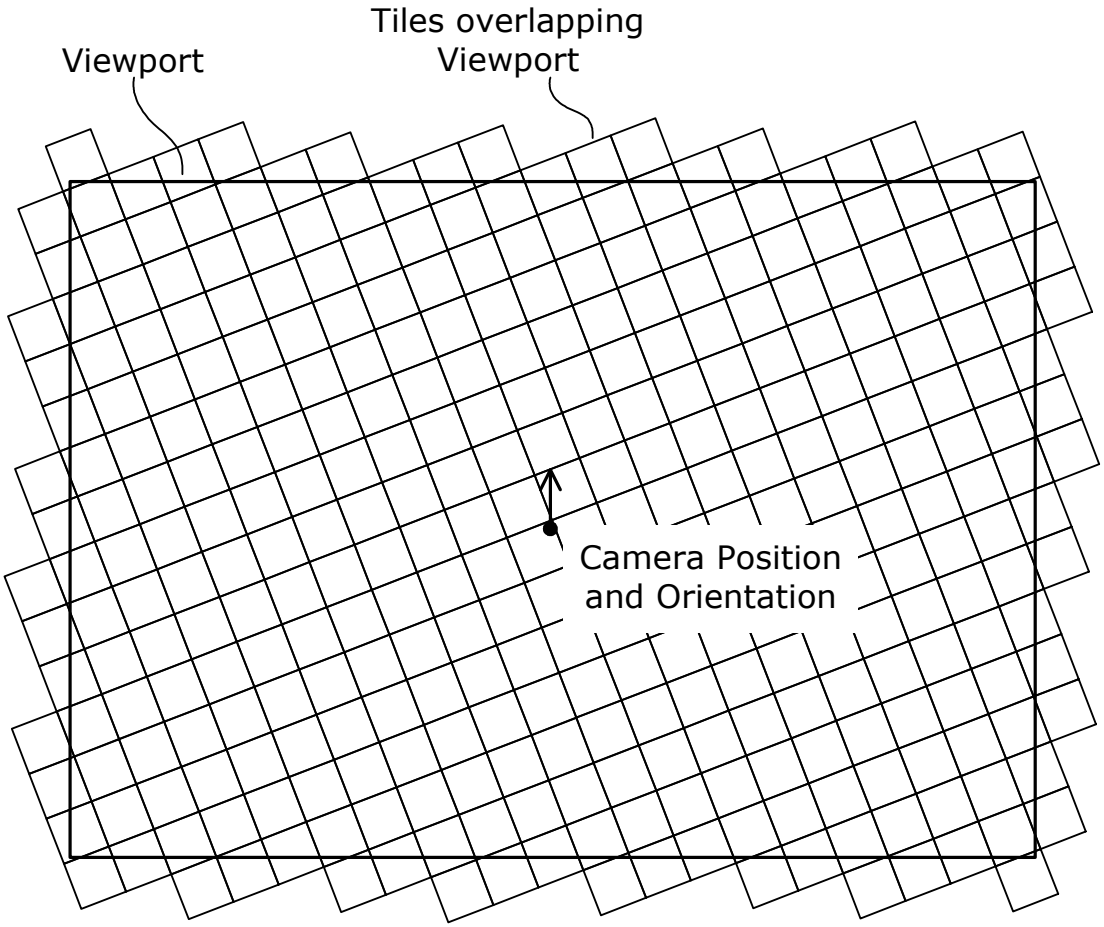
`G2DDrawLayer` is the most important function of the API. This function renders the data passed in via **layer*. The *aSortBuffer* parameter should point to a block of workspace memory that will be used to sort the tiles by material. This buffer must have a maximum size of 6 bytes times the number of tiles that fit in the viewport.

This function operates by first determining the minimal set of tiles which overlap the viewport (see "Figure 2–1 Example of G2DDrawLayer Functionality on page 8"). These tiles are sorted in order of material, and all tiles of the same material are rendered together.

`G2DInitSprite` sets up the texture coordinates of a sprite prior to rendering the sprite.

`G2DDrawSprite` renders a single sprite. It should be noted, however, that the main focus of the G2D API is not rendering sprites, but rather rendering layers of tiles. Therefore, you may prefer to write your own specific sprite routines, keeping in mind that they may need to be optimized (for example, for rendering multiple sprites of the same type).

Figure 2–1 Example of G2DDrawLayer Functionality



3 Graphics Requirements

The G2D API is optimized to render layers of tiles at high speed. It also allows the viewport to be rotated with respect to world coordinates, without introducing aliasing artifacts, and without paying the normal price for antialiasing. (The API achieves antialiasing for free by using the bilinear interpolation hardware.) However, there are certain rules to which you should adhere in order to achieve high performance and smooth antialiased graphics:

Everything must be arranged on power-of-2 boundaries.

To help caching and sorting performance, tiles used in the same part of the level design should be arranged within the same source texture map.

Graphics should join up smoothly, for which there are a couple of conditions (illustrated in Figure 3–1).

First, abrupt boundaries in graphics data should not occur at the edge of tiles. Tiles should be designed so that abrupt boundaries occur at least one pixel in from the edge of the tile. In other words, pixels on the border of tiles should be the same color as, or at least a close color to, any tile to which it will be adjacent in the final layer map. There may be more than one adjacent tile in the level design, so colors must be matched for all combinations of potentially adjacent tiles.

Second, tiles must be arranged in the source texture map in such a way that they join up with each other, or at least so that there is one extra pixel around the edge of any tiles. This is necessary because the bilinear interpolation hardware reads an extra pixel outside the tile in all directions. In practice, this means we may need to duplicate certain tiles in order to make things join up properly, but this is a small price to pay for antialiased rotations.

Figure 3–1 Layout Restrictions