Pixel Packing Monsters

Assuming you are familiar with how Color and Pixels are different in the NewTOAPIA library, you may be wondering how it is that you can use this information to draw pixel maps, and display things on the the screen. This document describes Pixel Arrays, and other components that help store, retrieve, access, and otherwise manipulate pixels and colors.

As a brief refresher, we have the ColorRGBA structure which represents color information in the NewTOAPIA libraries.

public struct ColorRGBA

{

public float r, g, b, a;

}

The ColorRGBA structure has many different uses, including being a base structure used to facilitate conversion between different color models. This structure takes up 16 bytes per individual color value. This is too much information to store for individual pixels, so pixels are typically stored in a smaller data structure, such as the BGRb data structure.

public struct BGRAb : IPixelBGRA<byte>

{

byte blue;

byte green;

byte red;

byte alpha;

}

bThis data structure uses bytes to represent the individual components, and thus takes up 4 bytes per pixel, rather than 16. Besides taking fewer bytes, it also fixes the range of the individual components to be from 0 to 255. Knowing the range of the pixels is important when doing any image processing.

Pixels are typically stored in some type of array, of dimension one, two, or three. As simple data structures, they can be used in any array just like any other structures. But, pixels are different than typical data structures in that we know they are used to render images. We also know that an extremely common usage pattern is to use pixels in a two dimensional array, typically known as a Bitmap. In this document, we refer to a two dimensional array of Pixel values as a Pixel Array.

The Pixel Array is a specialized array that gives us both individual element access as well as providing some convenience functions. One example of this is that Pixels can be stored in a specific orientation, such as top to bottom, or bottom to top. If it were just a basic two dimensional array, this orientation would not be relevant, but for images it is.

We also need to know information such as ‘Stride’. Stide is typically the number of elements that must be traversed to moved from row to row in the array. In most cases the stride is the same as the width value of the Array, but it can be different. One common example is when you have a Pixel Array, let’s say it is 640x480, and you want to describe a subsection of that array, say an area that is 320x240. In this case, the subsection would have a width of 320 a height of 240, and a stride of 640.

Describe PixelArray<T>

Describe PixelAccessor<T>