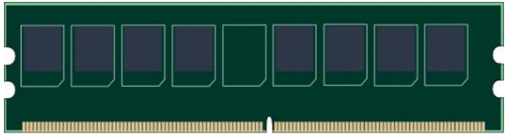
# Introduction to Framebuffers & Pixel Buffers

A speed run through containers


http://www.gameplayer.com.au/gametube-mario-64-tas-speed-runs/

# The basic concept (A Memory Buffer)



Main (CPU)Memory


Buffers store data in memory.

**Framebuffers** and **pixel buffers** are specialized buffers for storing visual data.

Data is nothing without meaning.

# Pixel Buffer vs Framebuffer

Traditionally, Pixel Buffers and Frame Buffers were just ***buffers***.

# Pixel Buffer vs Framebuffer

The variations depend on the toolkit.

### Pixel Buffer

- Stores Data about Pixels
- Transformations
- Drawing Operations
- CPU (Software) Rendering
- Not connected to an OpenGL Texture

### Framebuffer

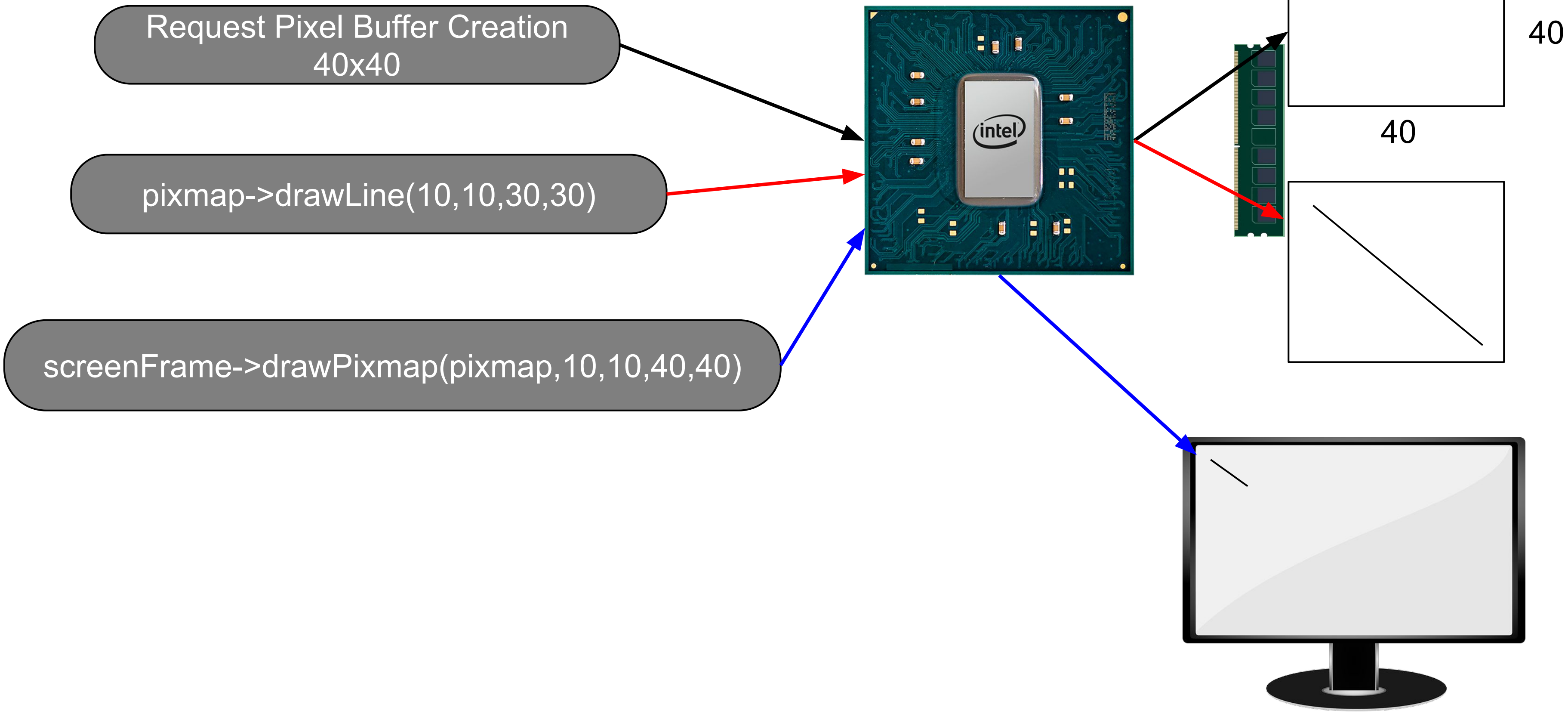
- Stores Data about Pixels
- Transformations
- Drawing Operations
- GPU (Hardware) Rendering
- Connected to an OpenGL Texture

But for now - add these general usage variations.

https://www.opengl.org/wiki/Pixel_Buffer_Object

https://www.opengl.org/wiki/Framebuffer_Object

# Software Rendering



# Hardware Rendering

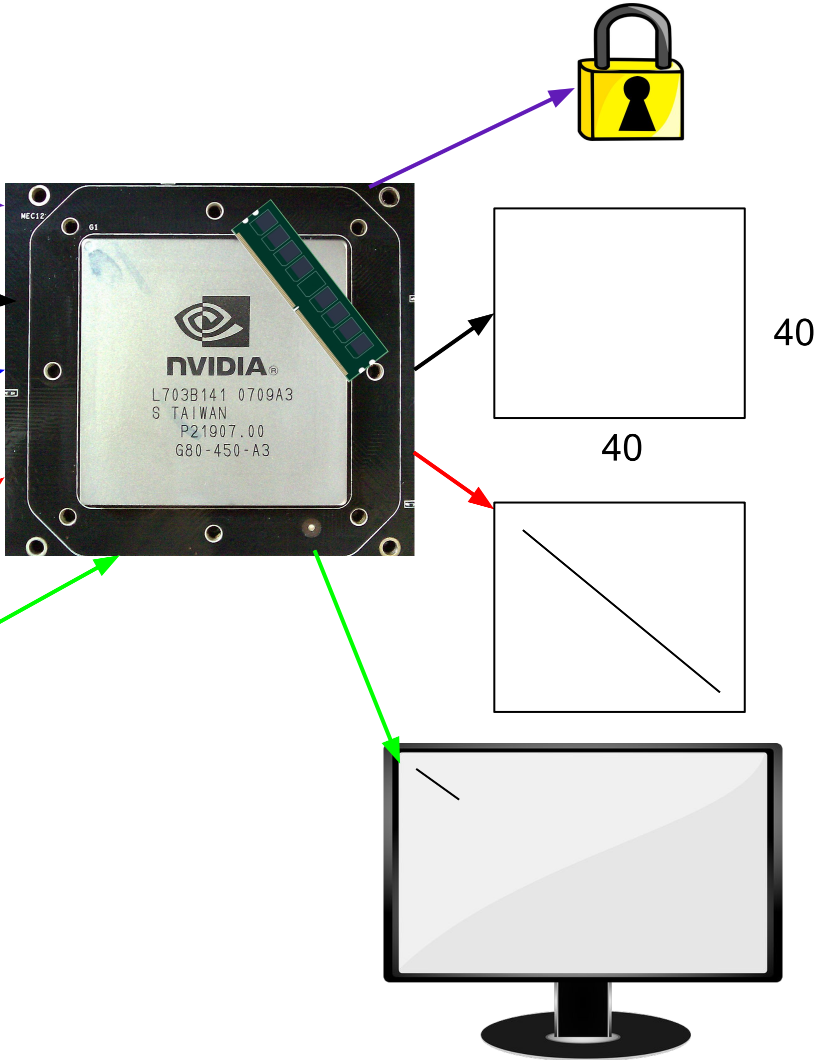
Request Opengl Context

Request Pixel Buffer Creation
40x40

Create vector describing line
**lineVector**[10,10,30,30]

frameBuffer->drawVector(10,10,lineVector)

frameBuffer->swapBuffer()



Pros and Cons to both

Hardware Rendering is not always available

Software Rendering is slow

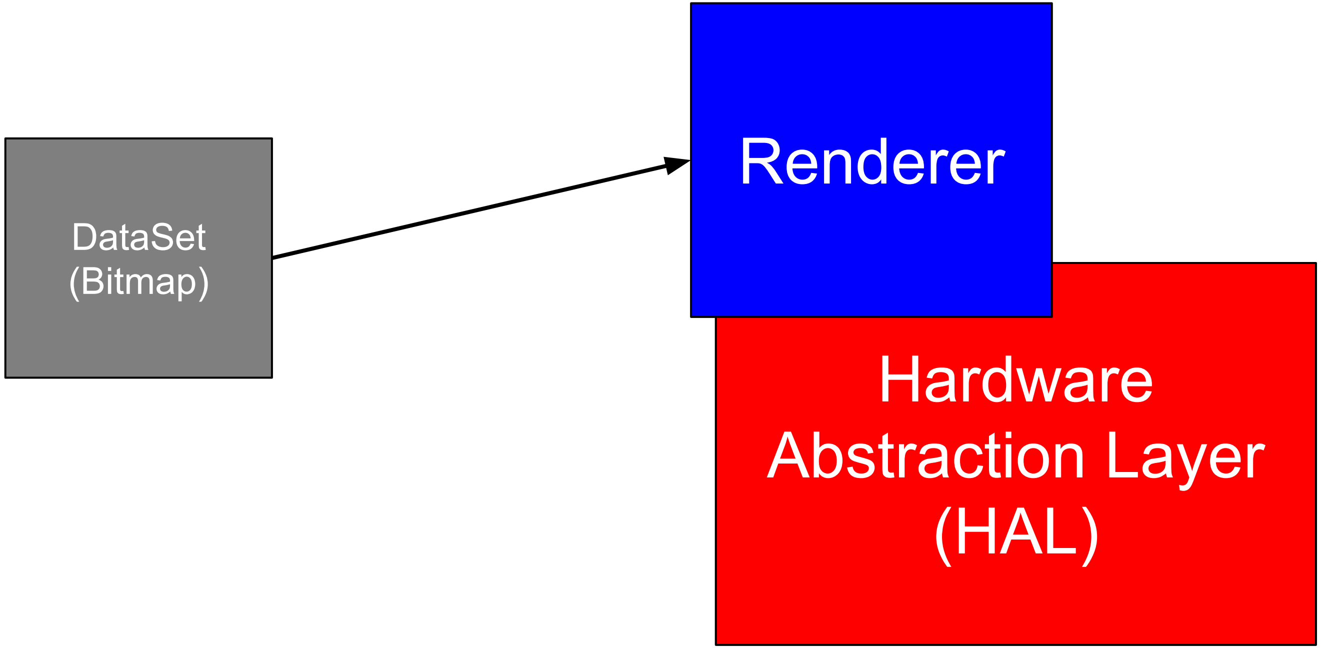
# Takeaway

More setup for HW Rendering

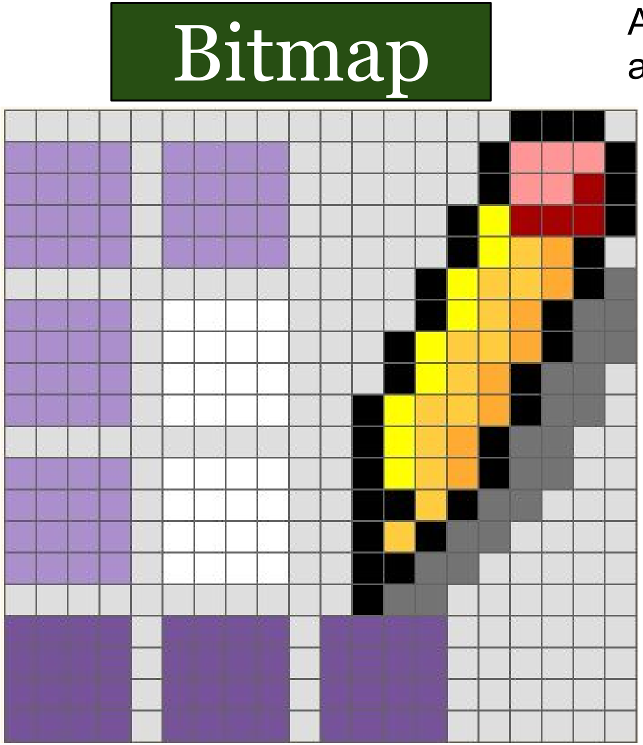
Without threading SW rendering almost doesn’t work

There is more than one type of memory (CPU & GPU)

# 3 Basic Elements Needed to Draw an Image



# What is a bitmap?




A collection of bits (now more commonly bytes) representing an image. The data is **formatted** (optimized) for the display.

**NOTE -** Bitmap is a **general term** and also a **file format**, they are not the same.

# Color Depth

| Bit-Depth | Number of Colors |
|---|---|
| 1 | 2 (monochrome) |
| 2 | 4 (CGA) |
| 4 | 16 (EGA) |
| 8 | 256 (VGA) |
| 16 | 65,536 (High Color, XGA) |
| 24 | 16,777,216 (True Color, SVGA) |
| 32 | 16,777,216 (True Color + Alpha Channel) |



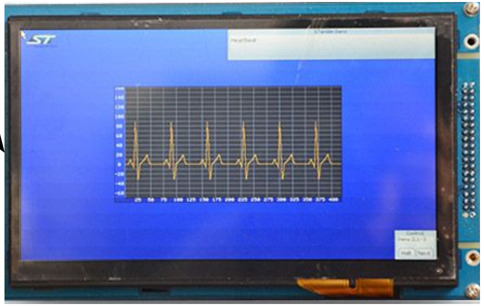
http://slideplayer.com/slide/6216504/

# Optimized for Display

| Bit-Depth | Number of Colors |
|---|---|
| 1 | 2<br>(monochrome) |
| 2 | 4<br>(CGA) |
| 4 | 16<br>(EGA) |
| 8 | 256<br>(VGA) |
| 16 | 65,536<br>(High Color, XGA) |
| 24 | 16,777,216<br>(True Color, SVGA) |
| 32 | 16,777,216<br>(True Color + Alpha Channel) |

http://slideplayer.com/slide/6216504/

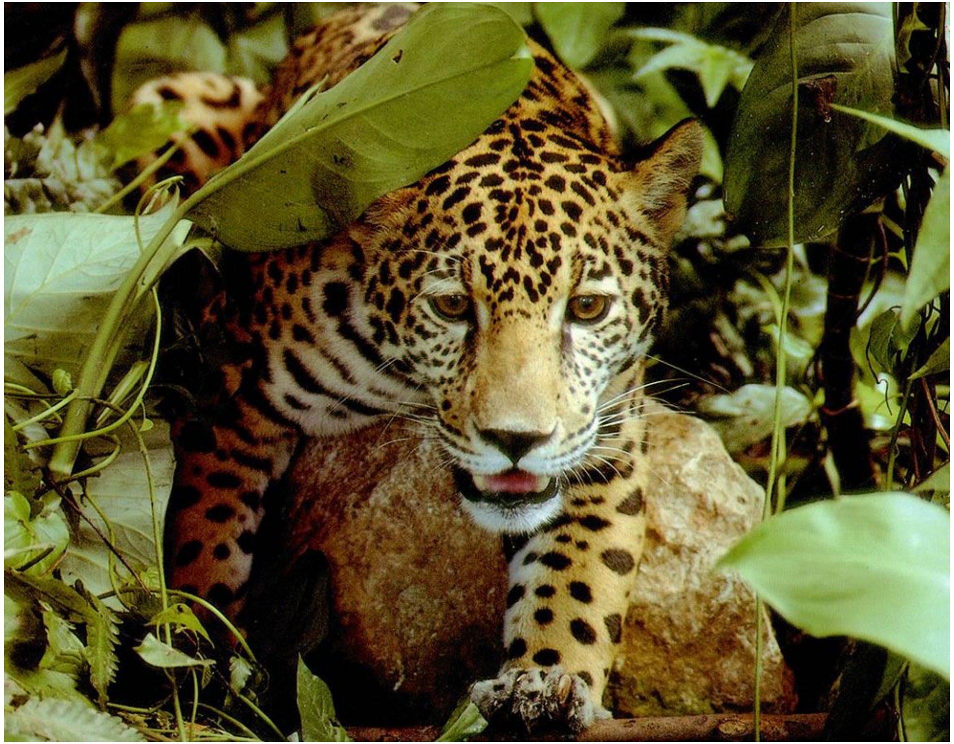




# Hardware limitations

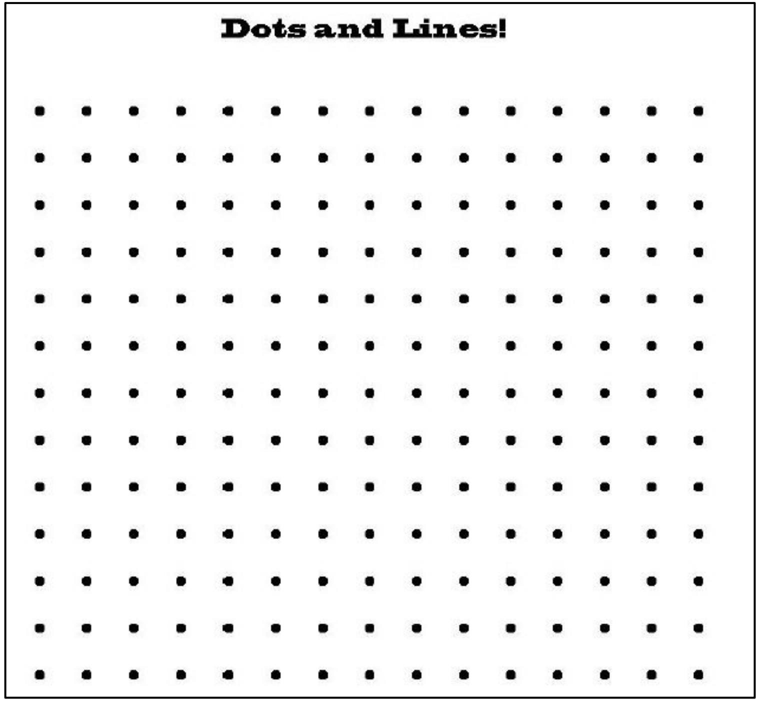

What if your hardware only allows 16-bit grafx?

# 32-bit RGBA



The more colors (depth) the better image?

# 32-bit RGBA (Wasted)



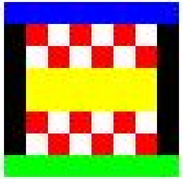

More colors (depth) **do not** guarantee better content!

# Real World Examples (BMPs)

The color table allows sharing color palette information only once.
Then the main data stores a reference to the (lookup) table.

4 Bits

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 0 | 1 | 4 | 1 | 4 | 1 | 4 | 0 |
| 0 | 4 | 1 | 4 | 1 | 4 | 1 | 0 |
| 0 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| 0 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| 0 | 1 | 4 | 1 | 4 | 1 | 4 | 0 |
| 0 | 4 | 1 | 4 | 1 | 4 | 1 | 0 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |



24 Bits

| | | |
|---|---|---|
| 0 | 000000 | |
| 1 | FF0000 | |
| 2 | 00FF00 | |
| 3 | 0000FF | |
| 4 | FFFFFF | |
| 5 | FFFF00 | |
| 6 | FF00FF | |
| 7 | 00FFFF | |
| 8 | FF0080 | |
| 9 | FF8040 | |
| A | 804000 | |
| B | 008080 | |
| C | 800000 | |
| D | 800080 | |
| E | 8080FF | |

0100000 100001 000 1000 1100

| R | R | R | R | R | R | R | R | G | G | G | G | G | G | G | G | B | B | B | B | B | B | B | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

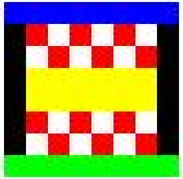
https://msdn.microsoft.com/en-us/library/windows/desktop/ms536393(v=vs.85).aspx

# Real World Examples (Different Formats)

The color table allows sharing color palette information only once. Then the main data stores a reference to the (lookup) table.

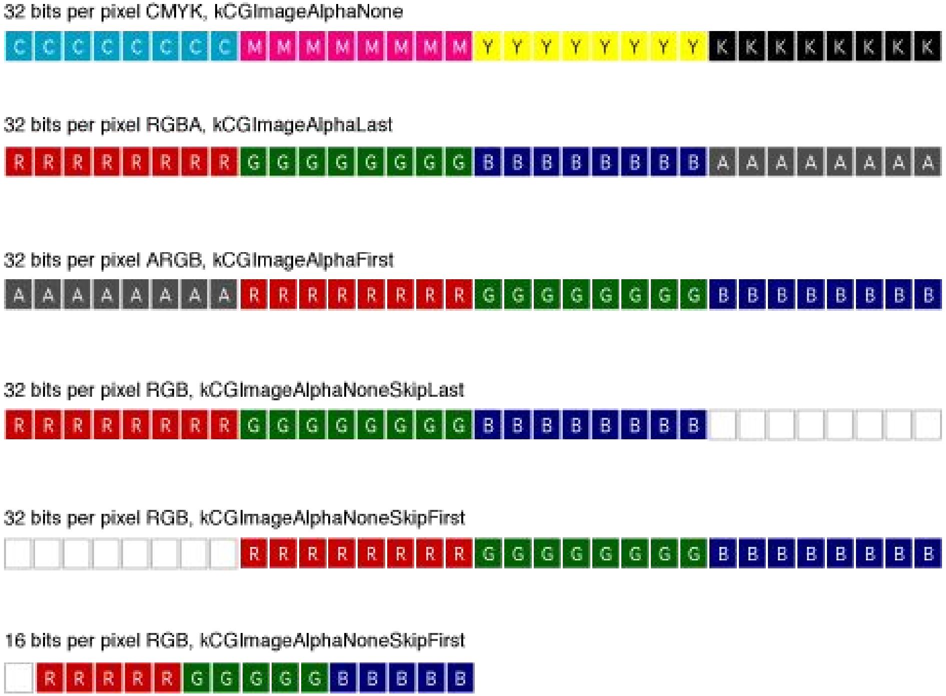
4 Bits

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 0 | 1 | 4 | 1 | 4 | 1 | 4 | 0 |
| 0 | 4 | 1 | 4 | 1 | 4 | 1 | 0 |
| 0 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| 0 | 5 | 5 | 5 | 5 | 5 | 5 | 0 |
| 0 | 1 | 4 | 1 | 4 | 1 | 4 | 0 |
| 0 | 4 | 1 | 4 | 1 | 4 | 1 | 0 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |



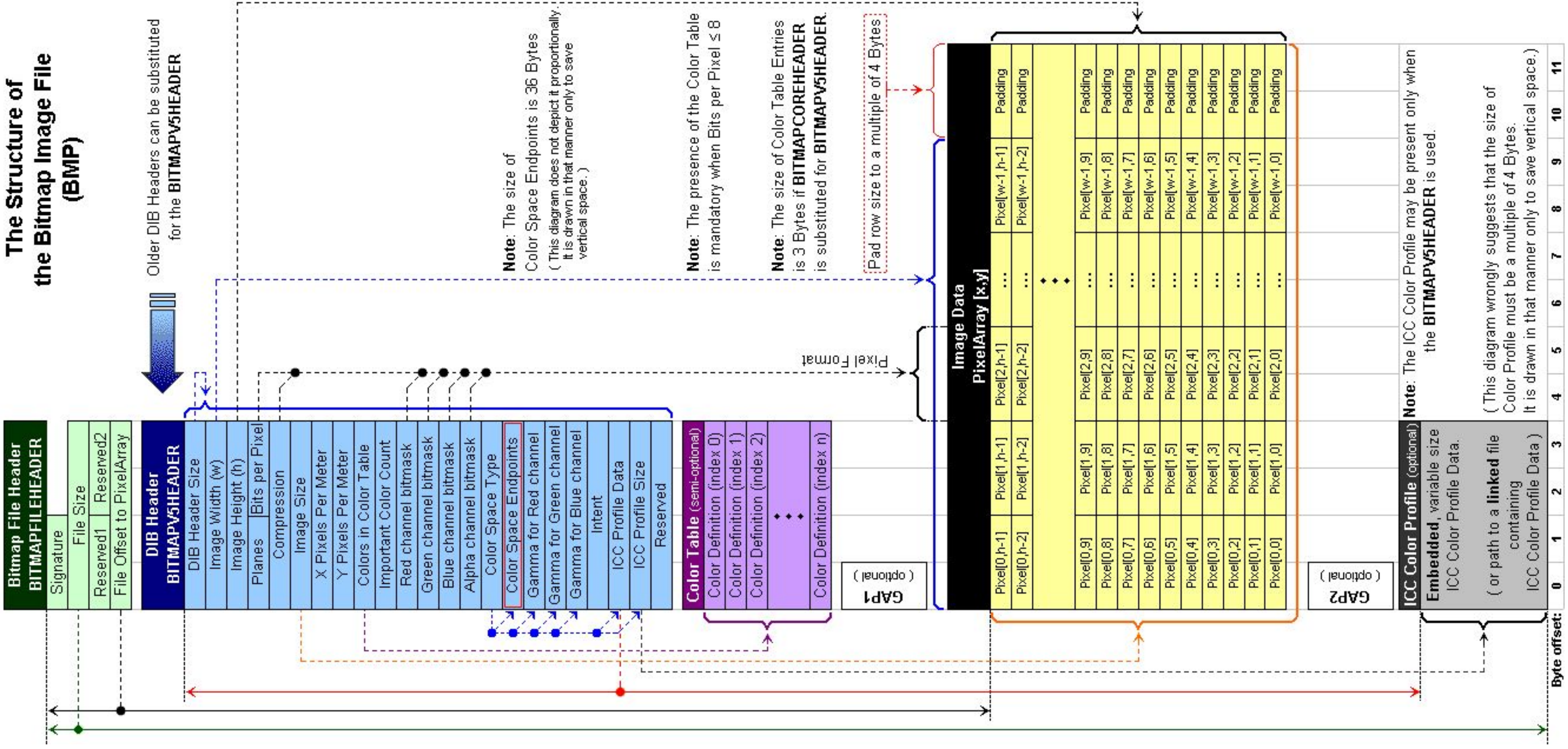
24 Bits

| | |
|---|---|
| 0 | 000000 |
| 1 | FF0000 |
| 2 | 00FF00 |
| 3 | 0000FF |
| 4 | FFFFFF |
| 5 | FFFF00 |
| 6 | FF00FF |
| 7 | 00FFFF |
| 8 | FF0080 |
| 9 | FF8040 |
| A | 804000 |
| B | 008080 |
| C | 800000 |
| D | 800080 |
| E | 8080FF |




https://msdn.microsoft.com/en-us/library/windows/desktop/ms536393(v=vs.85).aspx

# Real World Examples (Bitmaps - Technical)



File Formats store data differently

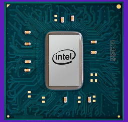
Each format has strengths and weaknesses

Most images won’t require RGBA

# Takeaway

# Rendering (Raster)

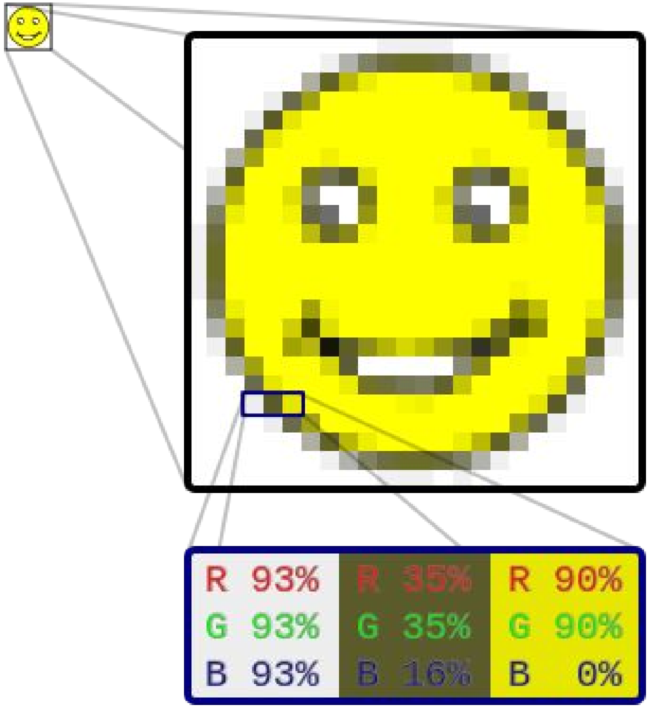
**Traditionally**



Performed by a CPU
(Before GPUs were common)

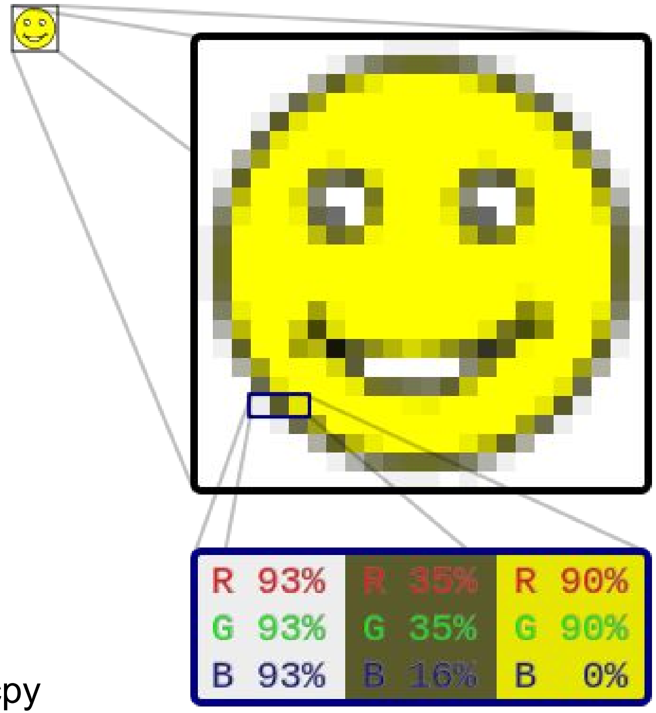
Had scan lines (rake)

Pixel by Pixel, Line by Line

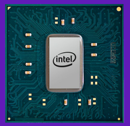



https://en.wikipedia.org/wiki/Raster_graphics

# Rendering (Raster)




**<u>Now more often used in</u>**



Software Rendering
(Pixmap Rendering)

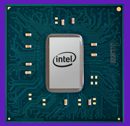
Printers (Raster Image Processor)

Copy one raster image into another is similar to a memcpy operation. New data replaces the old data.

https://en.wikipedia.org/wiki/Raster_graphics

# Rendering (Raster)

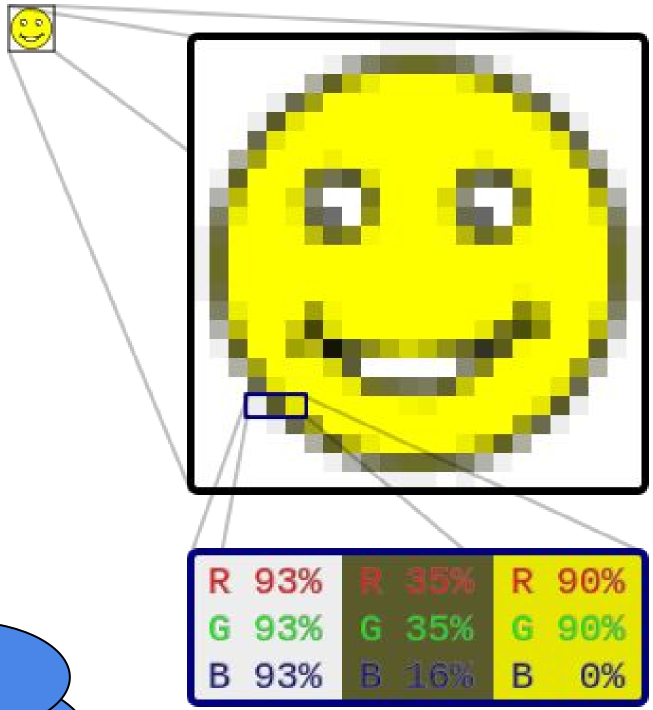
**<u>Now more often used in</u>**



Software Rendering
(Pixmap Rendering)

Printers (Raster Image Processor)

Alternatives?



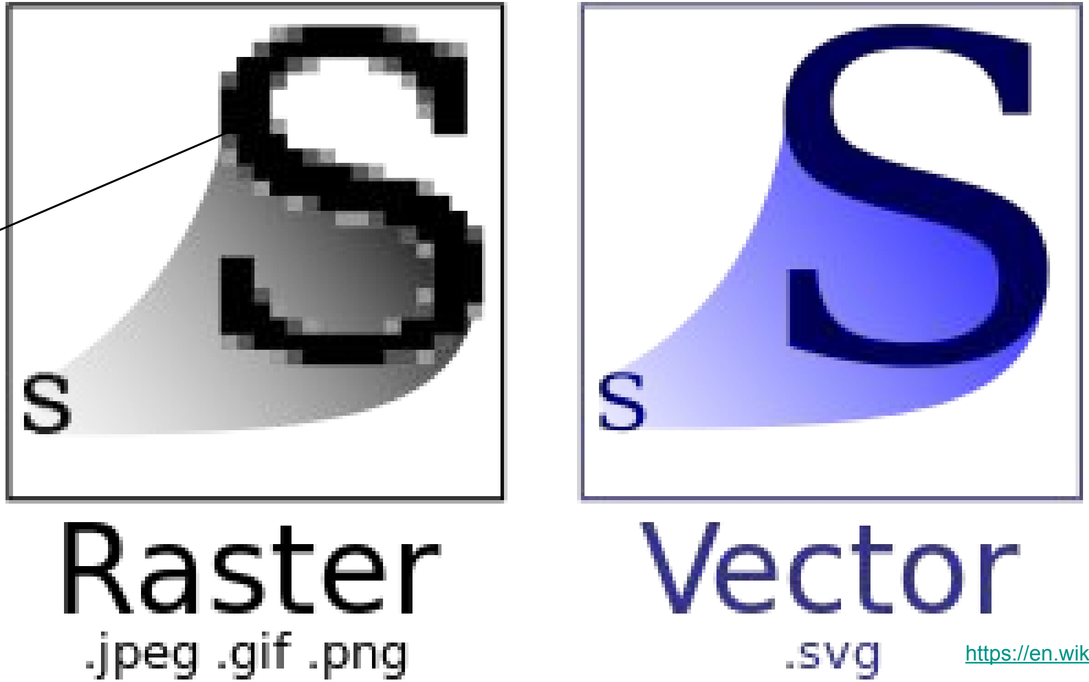

https://en.wikipedia.org/wiki/Raster_graphics

# Scalable Vector Graphics

SVG takes a different approach to rendering. Instead of drawing **pixel by pixel** SVG renderers **draw shapes**.

Raster images (made of pixels) can be scaled but can only replicate existing pixel data.

‘Pixelation’ Occurs

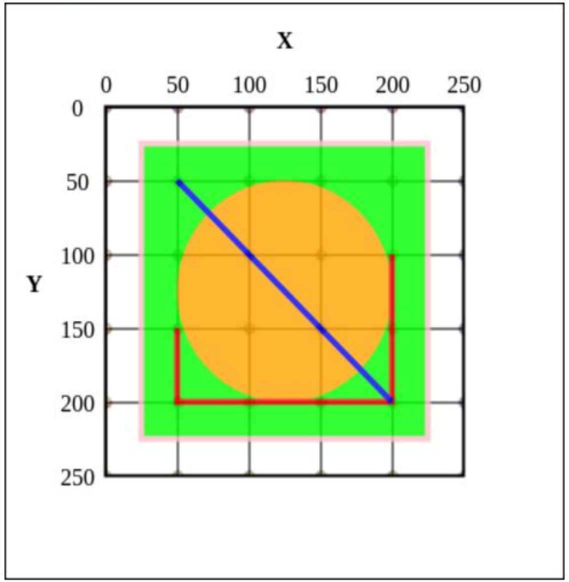



https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

# Scalable Vector Graphics (Internal)

An SVG document can define components including shapes, gradients etc., and use them repeatedly. SVG images can also contain raster graphics, such as PNG and JPEG images, and further SVG images.

### Example [ edit ]




This code will produce the shapes shown in the image (excluding the grid):

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="25" y="25" width="200" height="200" fill="lime" stroke-width="4" stroke="pink" />
  <circle cx="125" cy="125" r="75" fill="orange" />
  <polyline points="50,150 50,200 200,200 200,100" stroke="red" stroke-width="4" fill="none" />
  <line x1="50" y1="50" x2="200" y2="200" stroke="blue" stroke-width="4" />
</svg>
```

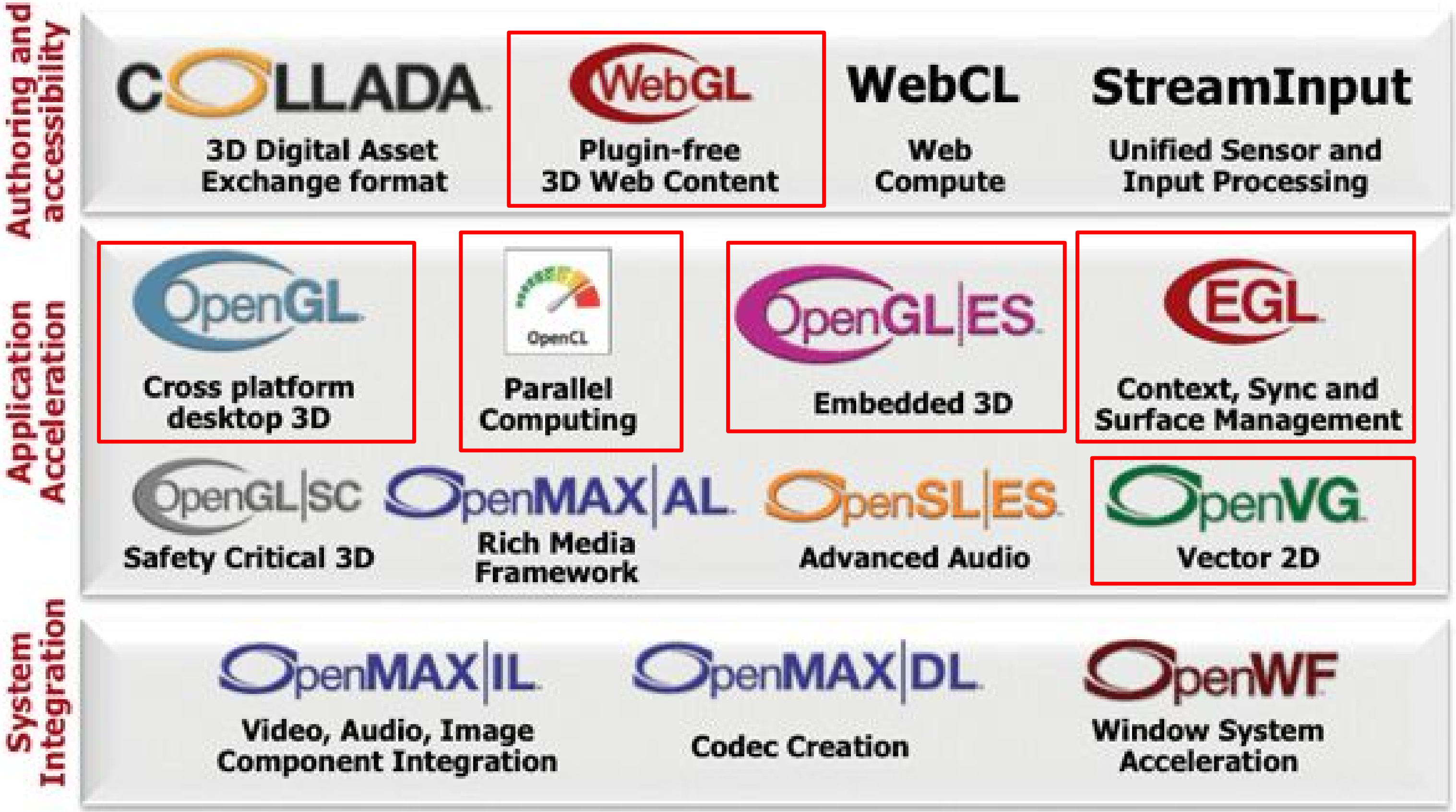
## Optimization

```
<svg xmlns="http://www.w3.org/2000/svg"
  version="1.1" width="100" height="100"
  shape-rendering="optimizeSpeed">
```

shape-rendering: geometricPrecision:

shape-rendering: optimizeSpeed

# Hardware Rendering (HAL)



# Pixel Buffers in Qt

## QPixmap Class

The QPixmap class is an off-screen image representation that can be used as a paint device.

| | |
|---|---|
| Header: | #include <QPixmap> |
| qmake: | QT += gui |
| Inherits: | QPaintDevice |
| Inherited By: | QBitmap |

## QImage Class

The QImage class provides a hardware-independent image representation that allows direct access to the pixel data, and can be used as a paint device.

| | |
|---|---|
| Header: | #include <QImage> |
| qmake: | QT += gui |
| Inherits: | QPaintDevice |

# Framebuffers in Qt

## QOpenGLFramebufferObject Class

The QOpenGLFramebufferObject class encapsulates an OpenGL framebuffer object.

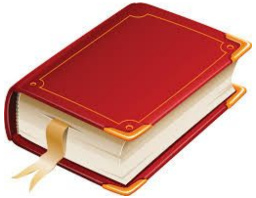
| | |
|---|---|
| Header: | #include <QOpenGLFramebufferObject> |
| qmake: | QT += gui |
| Since: | Qt 5.0 |

## QQuickFramebufferObject Class

The QQuickFramebufferObject class is a convenience class for integrating OpenGL rendering using a framebuffer object (FBO) with Qt Quick.

| | |
|---|---|
| Header: | #include <QQuickFramebufferObject> |
| qmake: | QT += quick |
| Since: | Qt 5.2 |
| Inherits: | QQuickItem |

# Suggested Reading



QML Book (Chapter 15) C++ Integration
http://qmlbook.github.io/en/ch15/index.html

http://doc.qt.io/qt-5/topics-graphics.html

# Live Coding Demo

Advanced Input Forwarding