1. **请指出“1080p”的意义？**

**1920 x 1080**

1. **请解释颜色的本质及其数字记录原理，并说出几个你所知道的色域。**

颜色原理：颜色的本质是一种光波（电磁波，波长介于380 – 760nm，可被人类看见的波段）它的存在是因为有三个实体：光线、被观察的对象以及观察者。人眼是把颜色当作由被观察对象吸收或者反射不同波长的光波形成的。例：红色物体吸收了其他波长的光，将红色的光反射出来

数字图像：**数字图像处理是指将图像信号转换成数字信号并利用计算机对其进行处理的过程。**图象数字化是电脑图象处理最基本的步骤，其意义就在于把真实的图象，转变成电脑所能接受的格式，也就是一连串特定的数字。常见的扫描仪就是这个过程，通常这个数字化的过程还可以分为“采样”与“量化”处理两个步骤。其中“采样”的结果就是通常所说的图象分辨率，而“量化”的结果则是图象所能容纳的颜色总数。

采样处理：采样的意义就是要使用多少点(像素点)来表示一张图象，例如：一幅640\*480的图象是由307200个点所组成，当然，想要有更清楚的图象质量，就得使用更多的点，来表示图象，也就是让这幅图象拥有较高的分辨率。

量化处理：量化的意义是指要使用多大范围的数值(颜色数)，来表示图象采样之后的每一个点。这个数值范围包含了图象上所能使用的颜色总数，eg:以4个Bits存储一个点，就表示图象只能有16中颜色。数值范围越大，表示图象可以拥有更多的颜色，自然可以产生更为逼真的图象效果。

图像数据存储：位映射、向量处理

1. 位映射：可以将图象的每一点数值存放在以字节为单位的矩阵里，比如：当图象是单色时(黑白色)，一个字节(8位)可存放8点(像素点)图象数据，16色图象则是以一个字节存2点，256色图象则是一个字节存储1点。这种存储方式适合内容复杂的图象。
2. 向量处理：只记录图象内容的轮廓部分，而不存储图象数据的每一点，比如：一个圆形图案只要存储圆心的坐标位置和半径长度，还有圆形边线及内部的颜色，适合存储商用图表和工程设计图。
3. **请解释“矢量图”和“位图”的区别？**

位图：也叫做点阵图，删格图象，像素图，简单的说，就是最小单位由象素构成的图，缩放会失真。构成位图的最小单位是象素，位图就是由象素阵列的排列来实现其显示效果的，每个象素有自己的颜色信息，在对位图图像进行编辑操作的时候，可操作的对象是每个象素，我们可以改变图像的色相、饱和度、明度，从而改变图像的显示效果。举个例子来说，位图图像就好比在巨大的沙盘上画好的画，当你从远处看的时候，画面细腻多彩，但是当你靠的非常近的时候，你就能看到组成画面的每粒沙子以及每个沙粒单纯的不可变化颜色。

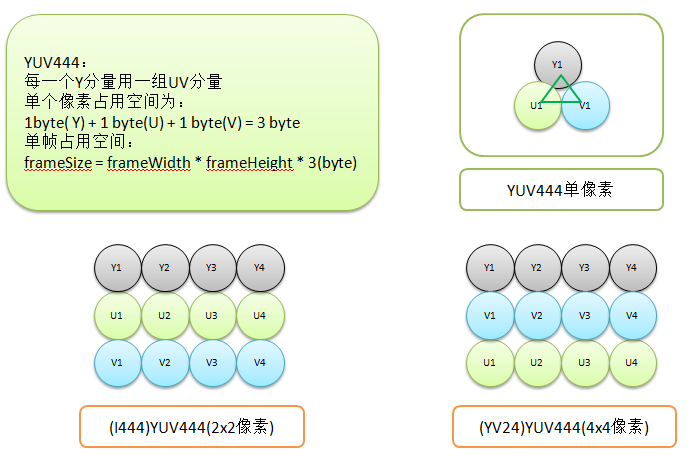
矢量图：也叫做向量图，简单的说，就是缩放不失真的图像格式。矢量图是通过多个对象的组合生成的，对其中的每一个对象的纪录方式，都是以数学函数来实现的，也就是说，矢量图实际上并不是象位图那样纪录画面上每一点的信息，而是纪录了元素形状及颜色的算法，当你打开一付矢量图的时候，软件对图形象对应的函数进行运算，将运算结果[图形的形状和颜色]显示给你看。无论显示画面是大还是小，画面上的对象对应的算法是不变的，所以，即使对画面进行倍数相当大的缩放，其显示效果仍然相同[不失真]。举例来说，矢量图就好比画在质量非常好的橡胶膜上的图，不管对橡胶膜怎样的常宽等比成倍拉伸，画面依然清晰，不管你离得多么近去看，也不会看到图形的最小单位。

区别：位图的好处是，色彩变化丰富，编辑上，可以改变任何形状的区域的色彩显示效果，相应的，要实现的效果越复杂，需要的象素数越多，图像文件的大小[长宽]和体积[存储空间]越大。

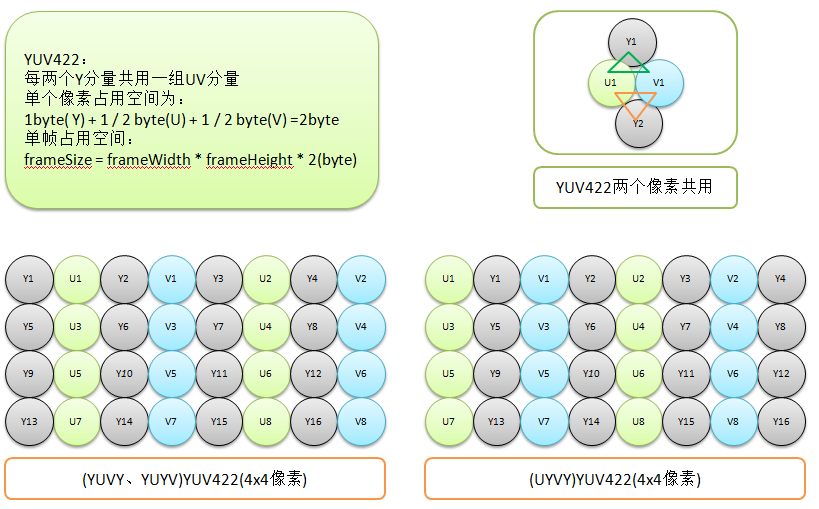
矢量的好处是，轮廓的形状更容易修改和控制，但是对于单独的对象，色彩上变化的实现不如位图来的方便直接。另外，支持矢量格式的应用程序也远远没有支持位图的多，很多矢量图形都需要专门设计的程序才能打开浏览和编辑。

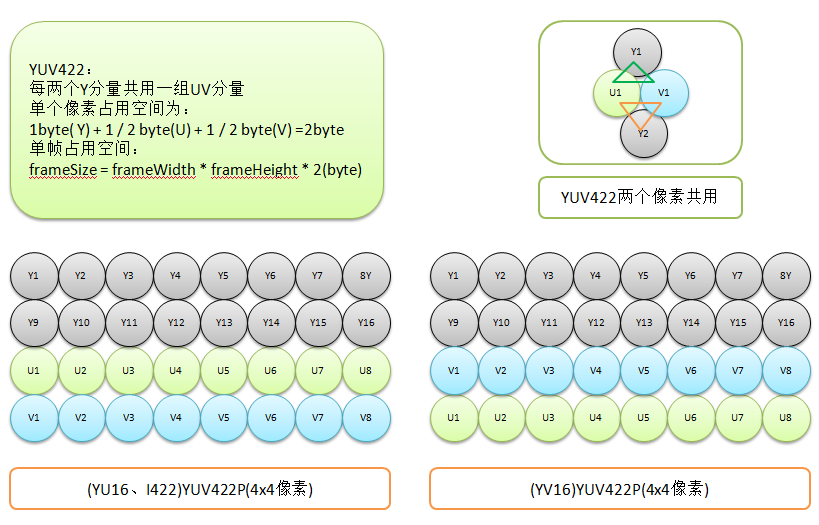
1. **视频分量YUV的意义及数字化格式？**

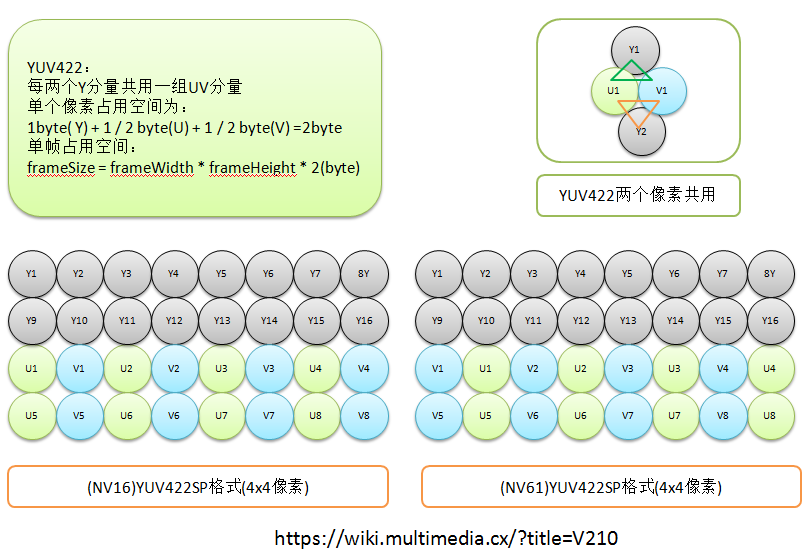
YUV444：



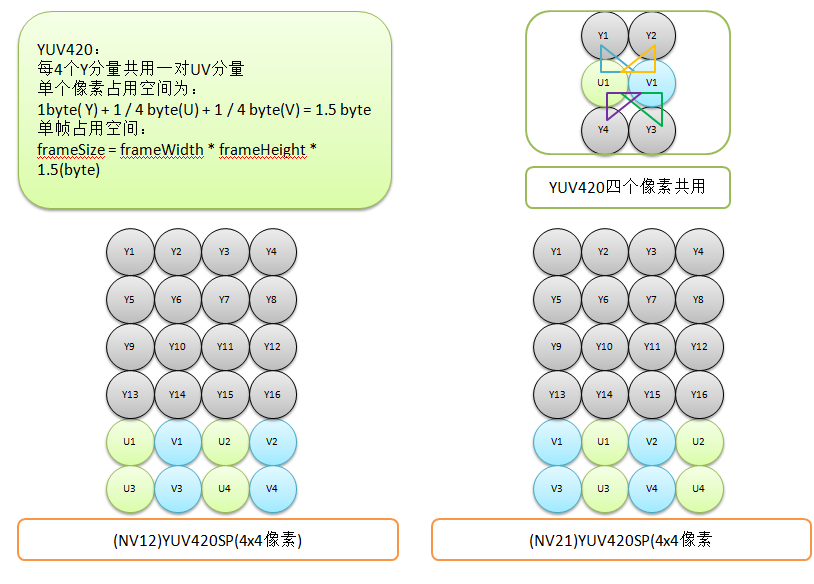
YUV422：

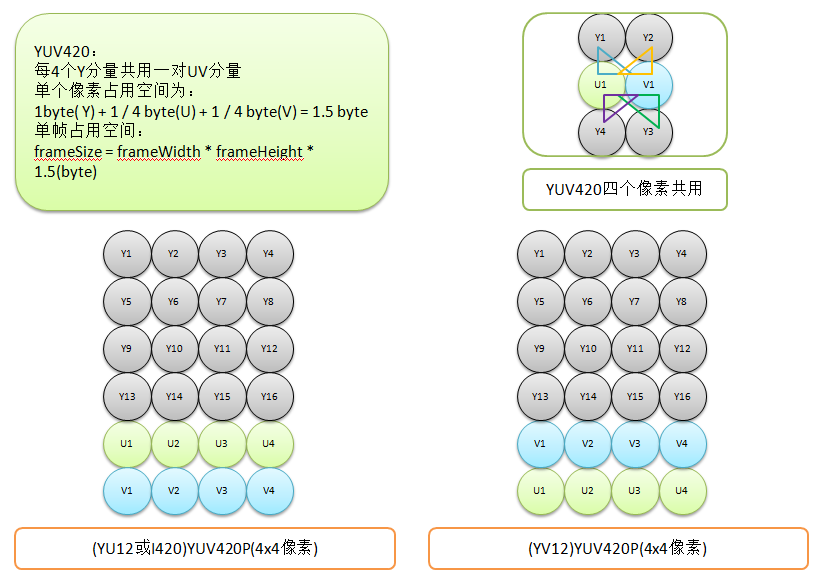






YUV420：





1. **RTP 打包模式？**

（1）Single NALU Packet（ 单一 NAL 单元模式） :

  即一个 RTP 包仅由一个完整的 NALU 组成 在一个RTP 包中只封装一个NALU，在本文中对于小于 1400字节的NALU 便采用这种打包方案。这种情况下 RTP NAL 头类型字段和原始的 H.264的 NALU 头类型字段是一样的.

（2）Aggregation Packet（ 组合封包模式） :

在一个RTP 包中封装多个NALU，对于较小的NALU 可以采用这种打包方案，从而提高传输效率。即可能是由多个 NAL 单元组成一个 RTP 包。 分别有4种组合方式: STAP-A, STAP-B, MTAP16, MTAP24.  那么这里的类型值分别是 24, 25, 26 以及27（3）Fragmentation Unit（ 分片封包模式FU）:

一个NALU 封装在多个RTP包中，在本文中，对于大于1400字节的NALU 便采用这种方案进行拆包处理。用于把一个 NALU 单元封装成多个 RTP 包。 存在两种类型 FU-A 和 FU-B. 类型值分别是 28 和 29.

**6、虚函数（virtual）可以是内联函数（inline）吗？**

[Are "inline virtual" member functions ever actually "inlined"?](http://www.cs.technion.ac.il/users/yechiel/c++-faq/inline-virtuals.html)

* 虚函数可以是内联函数，内联是可以修饰虚函数的，但是当虚函数表现多态性的时候不能内联。
* 内联是在编译期建议编译器内联，而虚函数的多态性在运行期，编译器无法知道运行期调用哪个代码，因此虚函数表现为多态性时（运行期）不可以内联。
* inline virtual 唯一可以内联的时候是：编译器知道所调用的对象是哪个类（如 Base::who()），这只有在编译器具有实际对象而不是对象的指针或引用时才会发生。

虚函数内联使用

#include <iostream>

using namespace std;

class Base

{

public:

inline virtual void who()

{

cout << "I am Base\n";

}

virtual ~Base() {}

};

class Derived : public Base

{

public:

inline void who() // 不写inline时隐式内联

{

cout << "I am Derived\n";

}

};

int main()

{

// 此处的虚函数 who()，是通过类（Base）的具体对象（b）来调用的，编译期间就能确定了，所以它可以是内联的，但最终是否内联取决于编译器。

Base b;

b.who();

// 此处的虚函数是通过指针调用的，呈现多态性，需要在运行时期间才能确定，所以不能为内联。

Base \*ptr = new Derived();

ptr->who();

// 因为Base有虚析构函数（virtual ~Base() {}），所以 delete 时，会先调用派生类（Derived）析构函数，再调用基类（Base）析构函数，防止内存泄漏。

delete ptr;

ptr = nullptr;

system("pause");

return 0;

}