软件 程序员 编程语言 编程 Java

关注者 15 被浏览 16,454

编程中代码转换为我们所见的图像、动作的原理是什么？

现在本人已知代码是通过一些程序软件转化为我们平常所见的图像、动作或是系统程序之类的，但是这种转化软件的原理是什么呢？比如游戏里一个跑步的动作，我们输入一些代码（好像是：move c……）在java里，那么java等这些程序又是如何运作的呢？还有这些编程软件最初的时候（没有编程软件的时候）又是怎样开发出来的呢？

3 个回答

默认排序

张若闲

人生如梦，岁月无情。蓦然回首，才发现人活着是一种心情。

+ 关注

2 人赞同了该回答

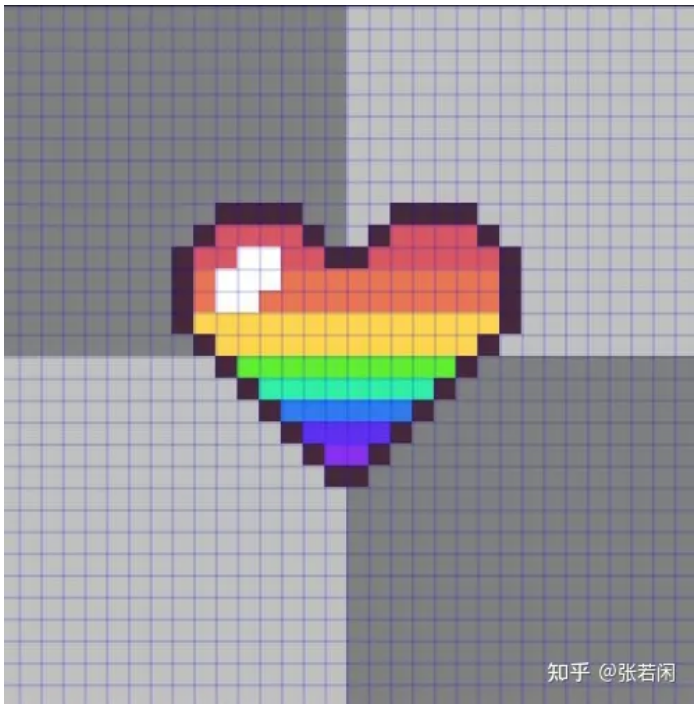
我们一点点的来，捋清楚。

首先，我不知道你有没有玩过，在本子上每一页上画小人，然后翻动本子就会感觉小人动了起来。这就是动画的基本原理，视觉暂留。由于我们连续的看到一连串的动作变换，然后就感觉真的动了起来。

显示器也是如此原理。

一、如何呈现出图片

我们先说说如何在屏幕上显示一个静态图。我们现在的屏幕，可以理解为密密麻麻的像素点矩阵。像是下面这样



每个点都可以表示一种颜色，最后整体呈现出来一副完整的图片。这个点越密集，我们看上去就会越清晰，反之就看着越粗糙。你看现在都会说屏幕分辨率是多少，实际上就是屏幕上点阵的密度问题。

什么又是像素点呢？可以理解为每一个像素点都是一个小灯泡，小灯泡亮起来就能看见，一堆小灯泡亮起来，就能看见一个平面的内容。当然，真实的屏幕并不是真的一堆灯泡，不过意思差不多意思。

程序呢，就控制着每一个点，点亮度和颜色；点呢，也可以通过二维坐标系唯一确定。程序上就会有一堆 [0,0] 白色 [0,1] 白色 [0,2] 黑色。只是点亮对应的像素就是了。

理论呢，就是这样，至于深层次如何实现的，怎么处理的，代码什么样，不再展开了。

二、如何呈现出动图

动画实际上就是视觉暂留的原理，在短时间内，切换多幅图呈现出来，这样就不会感觉是一副副的画了，而是一个连贯的动作。

上面说了通过像素矩阵可以绘制图片，那么我们就可以连续短时间内连续绘制多幅图片，就产生了快速翻小本本的感觉。

程序上步骤就是： 绘制图形 -> 绘制图形 -> 绘制图形 循环往复。

显示器接收到一组显示，显示了出来，然后有是一组显示，接着显示了出来。连贯起来，就感觉是画面动了起来。

三、代码如何转换

屏幕只负责展示，职责很单一。动画的逻辑，都是由程序控制。程序控制，也是控制屏幕画面不断变换。那么程序首先得能控制屏幕显示什么。

实际上，不同的屏幕显示的原理是不一样的，控制的方式也不一定一样，屏幕大小也不一样，清晰度也不一样。那对于那么多不一样，程序怎么展示呢？屏幕对于主机，提供了驱动程序，可以理解为一个屏幕配了一个翻译官。

主机这边的程序是一套一样的，屏幕显示那边是一套。中间加一个驱动程序，负责翻译程序和显示的关系。比如程序说，我要显示一幅图，这个指令到了驱动那里，驱动又翻译成屏幕认识的，每个像素点的颜色是什么。屏幕接收到了每个像素点的颜色，然后点亮对应的点。画面就出来了。

但实际程序中，翻译官可不止一位。

一般驱动程序对接着操作系统，操作系统是最底层的支撑，协调各个硬件驱动。在操作系统上，又有很多应用程序。这些应用程序可能都不是一种语言，可能有java、C等等。操作系统又给这些上层语言作为支撑，做翻译官。程序本身，也有翻译简化。最终呈现在开发者面前的，就是很简单的指令。

像是问题描述中的 指令 move啥的，实际上都是高级语言翻译简化之后的内容。比如js下面这段代码

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
```

这段代码，js已经做了封装处理，对于开发者就相对友好些。

第一步获取画布，就是规定在哪个地方画图

第二行是准备画笔，用ctx才能画图

第三行是使用什么颜色，定义了十六进制颜色

第四行就是画一个矩形，矩形四个参数：矩形顶点在屏幕上的 xy坐标，矩形的宽高。这四个值能够唯一确定矩形的位置和大小。

然后你就会在屏幕上看到一个矩形。

这是js呈现的语法，但是在幕后还有一堆看不见的翻译帮你执行着这一切。

最后

程序不是忽然间就是现在这样的，程序也经历了很长时间的发展。最开始的时候哪有屏幕，都是卡带。后来有了屏幕电视，也是模拟信号，也不是现在程序控制显示的样子。慢慢的发展，操作系统的不断完善，硬件设备的不断发展，才有了上层软件今天的样子。

所以说，不要问软件最初是怎么开发出来的，最早的程序没有屏幕。屏幕这种事，不是有程序才有的屏幕，而是有了屏幕后才有了对应控制屏幕的程序，程序员又觉得处理屏幕太麻烦，又抽出来作为单独控制屏幕的驱动程序。抽出来之后，其他人控制屏幕就容易了太多。但是这样还是麻烦，又让操作系统做了很多共性的工作，解放上层程序员不再去关心底层设备是啥，不用关心屏幕驱动是啥。

编辑于 2021-03-03 10:40

张砸锅
To Be a Better Coder
+ 关注

> **编程中代码转换为我们所见的图像、动作的原理是什么？**
> 现在本人已知代码是通过一些程序软件转化为我们平常所见的图像、动作或是系统程序之类的，但是这种转化软件的原理是什么呢？比如游戏里一个跑步的动作，我们输入一些代码（好像是：move c……）在java里，那么java等这些程序又是如何运作的呢？还有这些编程软件最初的时候（没有编程软件的时候）又是怎样开发出来的呢？

看了一下问题日志，提问题的人应该是一个设计师艺术家，隔行如隔山，所以不懂计算机世界里的奥妙。

对于计算机图形学我也只知皮毛，就简单说说我所知道的一些基本原理吧，不足之处有待行家指正。

先说说图像。计算机内部表示图像的方式简单的说可以分为两种，一种是光栅化的，一种是矢量化的。

光栅化，可以想象成一个很密的矩形网格罩在一幅图像上，只要把每个格的颜色表示出来了，那么整幅图也就表示出来了。所以，光栅化的图像在计算机内部的表示就是一堆颜色数值的二维矩阵，表示了每个图像点的颜色。这种表示方法好处是直观，人眼看到的就是颜色点颜色块，而且这种方式能直接对应到显示器的显示，因为绝大部分显示器是光栅化显示器，直接可以显示这类图像。缺点就是图像的缩放等几何变换会比较麻烦，而且很可能会损失源图像的精度。比如源图像是 200x200 个点，要放大成 400x400 个点，那原来的一个点需要变成现在的 4 个点，如果直接复制四份，那出来的不就是马赛克了，所以需要一些插值算法，甚至 AI 智能算法。

矢量化的图像，一般也叫做图形，是另一种思路，是用点、直线、曲线、填充等等形状元素来表示一幅图的。在计算机内部，表示的就是这些点、直线、曲线、填充等等的坐标以及相关参数。之所以叫做矢量化图性，是因为计算机里的坐标通常用矢量表示的。矢量化的好处是几何变换不失真，因为内部表示的是坐标数据，几何变换只要根据变换规则计算出新坐标位置就行，做各种计算是计算机的强项。缺点是不直观，因为人眼视觉感受的直接是颜色刺激，对于看到的东西并不会自动分解成几何形状，更不会分解出多重几何形状及颜色填充的叠加。

需要说明的是，对于矢量化的图性，最终要在显示器上展示出来，要做一次光栅化处理。前面说了，因为显示器一般是光栅化的。也由此可知，矢量变光栅是比较容易的，反过来，把任意一个光栅图像变成纯矢量图，并不是那么容易的。

以上是关于图像表示的简单介绍，现在再说说怎么让图像动起来。由于图像表示的方法不同，让图像动起来的方法也有所不同。

让光栅化图像动起来的结果，通常就是视频。其实也比较简单，和电影、电视的原理一样，视频是由一帧一帧的图像构成的，每一帧都是一幅完整的光栅化图像。对于一个 1080P 每秒 60 帧的视频，如果颜色深度是 16-bit，也就是 2 个字节，那么单独一帧就需要 1280 * 1080 * 2 = 2764800 字节，也就是 2.7 兆字节，一般一部电影 90 分钟，就需要 2764800 * 60 * 90 * 60 = 895795200000 字节，也就是 800 多G字节，可见数据量是很大的。所以对于视频，会有各种压缩技术，压缩后再存储，就能节约很大存储空间。也由此可见，做视频处理，对计算机的内存、外存需求量会非常大，对计算机的算力也要求很高。

让矢量化图像动起来的结果，通常叫做动画。由于矢量化图像的内部表示是形状元素，所以动画就是记录了这些形状元素的变形、变化过程。比如某个或某组形状元素，在时间点A时在什么坐标、什么状态，在时间点B时在什么坐标、什么状态。如果有过使用 Adobe Flash 做动画的经历，应该会更容易理解这个概念。动画在屏幕上展示的过程，实际上是计算机实时的把动画变成视频的过程。比如帧率是 60，那就是实时计算出每隔 1/60 秒时这些形状元素的位置及状态，把矢量图光栅化，把光栅化结果展示在屏幕上。

以上说的基本都是二维图形图像的情形，如果讨论到三维，叫图像不太合适了，一般叫做三维模型。

三维模型，应该说是光栅化和矢量化的结合体，另外，为了更好的模拟我们所处的真实世界，还增加了其他复杂的东东，比如各种特效。可以简单的理解为，三维模型中使用矢量化方式描述了各种对象的位置及形状，用光栅化的方式来表示贴图等元素，用算法表示各种特效。

三维的动画，就更复杂了。可以是视点视角不动，模型中的对象在动；也可以是模型中的对象都是不动的，但是视点视角在不停变化；还可以是模型对象和视点视角两者都在运动和变化。三维动画在屏幕上的展示，其基本原理还是，某个时间点时，把模型中矢量化的对象，在对应当前视点视角的视平面上做投影，然后把投影结果光栅化，然后把光栅化结果展示在屏幕上。

题主问：

> 比如游戏里一个跑步的动作，我们输入一些代码（好像是：move c……）在java里，那么java等这些程序又是如何运作的呢？

输入一些代码，或者说指令，为什么这些代码控制了动画动作，其实就是计算机怎么分析和解释这些代码的语义了。这其中的原理，可以简单解释一下。通常这里会用到一种叫脚本引擎的东东。当脚本引擎会接收到你输入的代码，首先要进行词法和语法分析，变出一种叫语法树的东东，然后对语法树可以直接解释执行，也可以编译成计算机内部的可执行代码后再执行。执行的结果，就是实现这些动画动作。比如 move 可以用来定义对象运动位置的坐标，从而使得该对象在动画中会移动到指定位置。至于脚本引擎是用 Java 还是其他编程语言编写的，就无关紧要了，只要能实现同样的功能就行。

> 还有这些编程软件最初的时候（没有编程软件的时候）又是怎样开发出来的呢？

至于这个，简单的说就是从无到有、从简单到复杂，一步步积累走过来的。

计算机最初是人造出来的，造的时候就设计好了怎么能够对它编程。CPU 能够直接执行的叫做机器指令，不同的 CPU 有不同的机器指令集，机器指令都是2进制数。由于2进制数写起来经常会太长，不宜人类读写，所以会用16进制数来替代。

用机器指令编程太困难了，需要人脑记住不同的数能让计算机做什么事。为了简化，对应每种机器指令，人们使用一个类似英文单词的助记符来表示它的功能，然后使用这种助记符来编程，这就是

汇编语言了。助记符和机器指令通常是一一对应的，而助记符对于计算机是不能直接执行的，所以需要一个叫汇编程序的东东，把由助记符编写的程序翻译成机器指令，这样计算机就可以直接执行了。

用汇编语言编程虽然方便一些了，但是不同 CPU 的计算机用的机器指令不一样，对应的汇编语言也不一样，同样的程序移植到一台新计算机上，就需要重新编写一遍，太不方便了。于是，人们发明了高级语言，高级语言的好处就是不依赖于底层的计算机硬件的指令系统，而更关注于程序所要实现的功能本身。但是同样的，高级语言的程序计算机是不能直接执行的，所以需要一种叫编译程序的东东，把高级语言程序翻译成机器指令，然后让计算机可以直接执行。

至于如何用上述的机器语言、汇编语言，或者是高级语言去编写程序，那就是程序员码农们专业范畴的事了。

发布于 2021-02-20 12:21

遇酒须倾
搞机者，认知计算/密码学

4 人赞同了该回答

代码作为程序的一部分而被执行时，先经过编译，确认程序能够执行，并且把我们写的这个具有“可读性”的程序转换成计算机能够识别的语言（就是0和1）。然后再把这个机器语言文件存放在外存中，等操作系统可以执行它的时候，转入到内存中，由CPU逐条读出来然后执行，就是你看到的结果了。

最初没有程序语言的时候，用有洞纸带代表0和1的机器语言（那个时候的计算机也比现在简陋多了，没这么多功能）让计算机读取执行。那个时候的计算机也只是单纯地负责计算而已。

但全是0和1的机器语言太繁琐，不具有可读性，所以有了汇编语言。

但计算机不能识别汇编——那就用0和1写一个程序，让计算机通过这个程序，把汇编转成机器语言。

再后来大家觉得汇编也繁琐，就发明了更高级的语言，然后用汇编写一个程序，让计算机通过程序把更高级的语言转换成机器语言。

再后来，程序语言发展得很完善了，大家就用这个非常高级的语言写了一个程序，来让这个高级语言自己编译自己——现在用的很多编程软件就是这样的。

发布于 2014-01-02 17:52