



## Screens & 2D Graphics: Crash Course Computer Science #23

Crash Course: Computer Science

<https://youtube.com/watch?v=7Jr0SFMQ4Rs>

<https://nerdfighteria.info/v/7Jr0SFMQ4Rs>

[PBS Digital Studios intro plays]

Hi, I'm Carrie Anne, and welcome to Crash Course Computer Science. This 1960 PDP-1 is a great example of early computing with graphics. You can see a cabinet-sized computer on the left, an electromechanical teletype machine in the middle, and a round screen on the right. Note how they're separated.

That's because text-based tasks and graphical tasks were often distinct back then. In fact, these early computer screens had a very hard time rendering crisp text, whereas typed paper offered much higher contrast and resolution.

The most typical use for early computer screens was to keep track of a program's operation, like values and registers. It didn't make sense to have a teletype machine print this on paper over and over and over again. That would waste a lot of paper, and it was slow.

On the other hand, screens were dynamic and quick to update, perfect for temporary values. Computer screens were rarely considered for program output, though. Instead, any results from a computation were typically written to paper or some other more permanent medium. But screens were so darn useful that by the early 1960s, people started to use them for awesome things.

[Crash Course intro plays]

A lot of different display technologies have been created over the decades, but the most influential and also the earliest were Cathode Ray Tubes, or CRTs.

These work by shooting electrons out of an emitter at a phosphor-coated screen. When electrons hit the coating, it glows for a fraction of a second. Because electrons are charged particles, their paths can be manipulated with electromagnetic fields. Plates or coils are used inside to steer electrons to the desired position, both left-right and up-down.

With this control, there are two ways you can draw graphics. The first option is to direct the electron beam to trace out shapes. This is called vector scanning. Because the glow persists for a little bit, if you repeat the path quickly enough, you create a solid image.

The other option is to repeatedly follow a fixed path, scanning line by line from top left to bottom right, and looping over and over again. You only turn on the electron beam at certain points to create graphics. This is called raster scanning. With this approach, you can display shapes and even text all made of little line segments.

Eventually, as display technologies improved, it was possible to render crisp dots onto the screen, also known as pixels. The liquid crystal displays, or LCDs, that we use today are quite a different technology, but they use raster scanning, too, updating the brightness of little tiny red, green, and blue pixels many times a second.

Interestingly, most early computers didn't use pixels, not because they couldn't physically, but because it consumed way too much memory for the computers of the time.

A two hundred by two hundred pixel image contains forty thousand pixels. Even if you use just one bit of data for each pixel, that's black or white - not grayscale - the image would consume forty thousand bits of memory.

That would have gobbled up more than half of a PDP-1's entire RAM, so computer scientists and engineers had to come up with clever tricks to render graphics until memory size has caught up to our pixel-licious ambitions.

Instead of storing tens of thousands of pixels, early computers stored a much smaller grid of letters, most typically eighty by twenty five characters. That's two thousand characters in total. And if each is encoded into eight bits using something like ASCII, it would consume sixteen thousand bits of memory for an entire screen full of text, which is way more reasonable.

To pull this off, computers needed an extra piece of hardware that could read characters out of RAM and convert them into raster graphics to be drawn onto the screen. This was called a character generator, and they were basically the first graphics cards.

Inside, they had a little piece of read-only memory, a ROM, that stored graphics for each character, called a dot matrix pattern. If the graphics card saw the eight-bit code for the letter "K", then it would raster scan its 2-D pattern onto the screen in the appropriate position.

To do this, the character generator has special access to a portion of the computer's memory reserved for graphics, a region called the screen buffer. Computer programs wishing to render text to the screen simply manipulated the values stored in this region, just as they could with any other data in RAM.

This scheme required much less memory, but it also meant the only thing you could draw was text. Even still, people got pretty inventive with ASCII art. People also tried to make rudimentary pseudo-graphical interfaces out of this basic set of characters, using things like underscores and plus signs to create boxes, lines, and other primitive shapes.

But the character set was really too small to do anything terribly sophisticated, so various extensions to ASCII were made that added new semi-graphical characters, like IBM's CP437 character set seen here, which was used in DOS.

On some systems, the text color and background color could be defined with a few extra bits. That allowed glorious interfaces like this DOS example, which is built entirely out of the character set you just saw.

Character generators were a clever way to save memory, but they didn't provide any way to draw arbitrary shapes, and that's important if you want to draw content like electrical circuits, architectural plans, maps, and, well, pretty much everything that isn't text.

To do this without resorting to memory-gobbling pixels, computer scientists used the vector mode made available on CRTs. The idea is pretty straightforward. All content to be drawn on the screen is defined by a series of lines. There's no text. If you need to draw text, you have to draw it out of lines. Don't read between the lines here; there's only lines. Got it? Alright, no more wordplay, I'm drawing the line here.

Let's pretend this video is a Cartesian Plane, two hundred units wide and a hundred tall, with the origin, that's the (0,0) point, in the upper left corner. We can draw a shape with the following vector commands which we've borrowed from the Vectrex, an early vector display system.

First, we reset, which clears the screen, moves the drawing point of the electron gun to (0,0), and sets the brightness of lines to zero. Then, we move the drawing point down to (50,50) and set the light intensity to one hundred percent. With the intensity up, now we can move to (100,50), then (60,75), and then back to (50,50). The last thing to do is to set the light intensity back to zero percent.

Cool, we've got a triangle. This sequence of commands would



## Screens & 2D Graphics: Crash Course Computer Science #23

Crash Course: Computer Science

<https://youtube.com/watch?v=7Jr0SFMQ4Rs>

<https://nerdfighteria.info/v/7Jr0SFMQ4Rs>

consume about a hundred and sixty bits, which is way more efficient than keeping a huge matrix of pixel values.

Just like how characters were stored in memory and turned into graphics by a character generator, these vector instructions were also stored in memory and rendered to a screen using a vector graphics card. Hundreds of commands can be packed together sequentially in the screen buffer and used to build up complex graphics, all made of lines.

Because all these vectors are stored in memory, computer programs can update the values freely, allowing for graphics that change over time, animation!

One of the very earliest video games, *Spacewar!*, was built on a PDP-1 in 1962 using vector graphics. It's credited with inspiring many later games, like *Asteroids*, and even the first commercial arcade video game, *Computer Space*.

1962 was also a huge milestone because of Sketchpad, an interactive graphical interface that offered computer-aided design, called CAD software today. It's widely considered the earliest example of the complete graphical application, and its inventor, Ivan Sutherland, later won the Turing Award for this breakthrough.

To interact with graphics, Sketchpad used a recently invented input device called a light pen, which was a stylus tethered to a computer with a wire.

By using a light sensor in the tip, the pen detected the refresh of the computer monitor. Using the timing of the refresh, the computer could actually figure out the pen's position on the screen. With this light pen and various buttons on a gigantic computer, users could draw lines and other simple shapes.

Sketchpad could do things like make lines perfectly parallel, the same length, straighten corners into ninety degree intersections, and even scale shapes up and down dynamically. These things that were laborious on paper, a computer now did with the press of a button.

Users were also able to save complex designs they created, and then paste them into later designs, and even share with other people. You could have whole libraries of shapes, like electronic components and pieces of furniture that you could just plop in and manipulate in your creations.

This might all sound pretty routine from today's perspective, but in 1962, when computers were still cabinet-sized behemoths, chugging through punch cards, Sketchpad and light pens were equal parts eye-opening and brain-melting.

They represented a key turning point in how computers could be used. They were no longer just number crunching math machines that hummed along behind closed doors. Now, they were potential assistants, interactively augmenting human tasks.

The earliest computer displays with true pixel graphics emerged in the late 1960s. Bits in memory directly "mapped" to pixels on the screen, what are called bitmapped displays. With full pixel control, totally arbitrary graphics were possible.

You can think of a screen's graphics as a huge matrix of pixel values. As before, computers reserve a special region of memory for pixel data, called the frame buffer. In the early days, the computer's RAM was used, but later systems used special high-speed video RAM, or VRAM, which was located on the graphics card itself for high-speed access. This is how it's done today.

On an eight-bit grayscale screen, we can set values from zero intensity, which is black, to two hundred and fifty-five intensity, which is white. Well, actually, it might be green or orange, as many early displays couldn't do white.

Let's pretend this video is a really low resolution bitmapped screen with a resolution of sixty by thirty-five pixels. If we wanted to set the pixel at (10,10) to be white, we could do it with a piece of code like this. If we wanted to draw a line, let's say, from (30,0) to (30,35), we can use a loop like so, and this changes a whole line of pixels to white.

If we want to draw something more complicated, let's say, a rectangle, we need to know four values: the x and y coordinates of its starting corner and its width and height.

So far, we've drawn everything in white, so let's specify this rectangle to be gray. Gray is halfway between zero and two hundred and fifty-five, so that's a color value of a hundred and twenty seven.

Then, with two loops, one nested in the other so that the inner loop runs once for every iteration of the outer loop, we can draw a rectangle. When the computer executes our code as part of its draw routine, it colors in all the pixels we specified.

Let's wrap this up into a "draw rectangle" function like this. Now, to draw a second rectangle on the other side of the screen, maybe in black this time, we can just call our rectangle drawing function. Voila!

Just like the other graphics schemes we've discussed, programs can manipulate pixel data in the frame buffer, creating interactive graphics. Pong time!

Of course, programmers aren't wasting time writing drawing functions from scratch. They use graphic libraries with ready-to-go functions for drawing lines, curves, shapes, texts, and other cool stuff. Just a new level of abstraction.

The flexibility of bitmapped graphics opened up a whole new world of possibilities for interactive computing, but it remained expensive for decades.

As I mentioned last episode, by as late as 1971, it was estimated there were around seventy thousand electromechanical teletype machines and seventy thousand terminals in use in the United States. Amazingly, there were only around one thousand computers in the U.S. that had interactive graphical screens.

That's not a lot, but the stage was set, helped along by pioneering efforts like Sketchpad and *Spacewar!* for computer displays to become ubiquitous, and with them, the dawn of graphical user interfaces, which we'll cover in a few episodes. I'll see you next week.

Crash Course Computer Science is produced in association with PBS Digital Studios. At their channel, you can check out a playlist of shows like *Gross Science*, *ACS Reactions*, and *The Art Assignment*.

This episode was filmed at the Chad & Stacey Emighloz Studio in Indianapolis, Indiana, and it was made with the help of all these nice people and our wonderful graphics team, Thought Cafe. Thanks for watching, and try turning it off and then back on again.