CMPS 10 Lecture Notes: Lecture 4 (1-14-2016)

- Joke about doing whole class raising hands.

- Last time in class...
  - Last time we talked about how we could communicate all of the information of lecture just by using left and right hand gestures
  - And then we talked about how you cold do that to see the entire video of the lecture.
  - So let's talk about that again. And we'll talk about a series of approximations.

- So, in 20th century, the principle of film: if you see a series of images, it looks like it is continuous
- FPS means Frames per second.
  - 24 frames/second is more or less what we can perceive..
  - So, if we see 24 pictures of the teacher per second, then we will be seeing the full lecture (ignoring voice for the moment).
  - So task is now reduced to showing how to capture a single frame.
  - So let's take a single frame. We cut the frame into pieces. A bunch of small 'squares'
  - But before we do that... let's jump back a bit.

- Now, imagine for a moment, you were someone who used to be able to see, but then you came blind in adult life
  - So you know what the world looks like, you just can't see any more.
  - AND you have strong powers of imagination. So you can create a scene in your brain if enough information is given to you.
  - So we engage in a dialogue in which there is a frame, and it needs to be described to us.
  - one way to do it would be ask someone what does the frame depict, in english. And then you would expect an english response, like "oh, its a guy giving a lecture" and then you could ask more questions "is there a whiteboard? etc."
  - And that would help form a picture
- OR, another thing you could do, is draw a rectangle representing your frame, say that the horizontal is the X axis, and the Y axis is the vertical.
  - Call the lower left hand corner of the frame (0,0), and then the lower right hand corner is (1,0) and the upper left hand corner (0,1) and and upper right hand corner is (1,1)
  - AND take all of the colors and put them in a spectrum between 0 and 1. So each number is assigned a value between 0 1 (e.g. 0.7329823 is a color).
  - Alternatively, you could have a 3D coordinate system, where each axis represents a different color
    - RGB representation: the idea is that any color can be expressed as a mixture between Red, Green, and Blue. You can create any color that you like by taking the appropriate amount of each.
    - And you can 'normalize' so that the total amount of green you take is a number between 0 and 1 (same with blue and red).
    - Also, note that we aren't saying percent. We can also capture luminosity.
    - So, points in space that correspond to the same proportion of color, you get the same hue, but different brightness.
  - So, going back to our frame, any given point in it, we need exactly three numbers (the amount of red, green, and blue).
  - And then we know the color inside there.
  - And so there is now a little bit of cheating going on. We used the word "point" for the frame, which implies an infinite amount of precision.
    - So then, instead of talking about a specific point, instead think of it as asking about a specific interval. (so, a 'cell' in the grid, etc.)
    - So now you aren't asking about a specific point about infinite precision.
    - And the smaller the interval, the more likely that it will just be a single color.
  - So, okay, let's just agree up front that we divide the scene into a grid.

  - And we assign a number to each cell in the grid.
  - So, let's say that our grid has 64 cells in it. (each cell will be the 'average color')
  - So to fully specify it, it means that we need 64 triplets (where each triplet is the R, G, B value).
  - But what if we took all of those grids, and cut them in half so now we need 4 times as many triplets (256), it is now a more accurate representation, because there are no squares that will need any averaging.
  - This is what pixels are.
  - So, you have the camera, and the camera measures the color of every pixel.
  - This is exactly what you have when you have a digital picture. There is a partition of the frame (i.e. what the sensor sees) into a grid. the partition is fixed. And what the sensor measures from the light is three numbers per pixel, and then it just literally writes those numbers using some amount of accuracy for the representation for each color.
  - The technical term: Color Depth. (Wikipedia Article)
  - Usually it is 16 or 24 bits. So it tells you to what accuracy you specify the color.

- So, let's take a color, say red, and it is between 0 and 1 the amount that we want.
  - and we do that same question and answer game from last time, getting it closer and closer to what type of red we are hoping for.
  - We keep on dividing the space in half by asking if it is in the L interval or the R interval.
  - We get closer and closer to the type of red we want.
  - So, let's draw another tree structure.
    * But this time, the leaves aren't the characters of the alphabet. The far left leaf is 0, the far right leaf is 1.
    * Also, note that if you are a computer scientist, you like to number things from 0.
    * So , with five levels, the final level would be numbered 4, and the number of leaves there is 2^level (or here, 2^4, or 16).
    * so each leaf is going to be evenly spaced by 1/16.
    * So now, if we have any real number in mind, we can specify it with any degree of accuracy, up to 1/16.
    * So, let's take 0.334. The first question is "less than a half or more than a half" (less). Then the second question is "less or more than a quarter" (more). Each question chooses which branch we go down.
    * And again, how much accuracy do we have?
    * The worst that could absolutely happen, after this dialogue of 4 questions, we would be able to know that the number we have in mind is SOMEWHERE between two 1/16
    * So the worst that can happen is that we are off by 1/16.
    * BUT, we could continue to play the game, to continue to reduce the uncertainty. One more round would make the accuracy down to 1/32. Another round after that would make it only to 1/64.
  - SO: 24 bit color depth per channel (R, G, and B would be three channels).
    * Means that there are three trees, one for Red, one for Green, one for Blue
    * And they all have depth 24.
    * Which means that, down at the bottom of the tree, there are two to the 24 little such dots, or two to the 24 leaves. Which means it is 2^10 * 2^10 * 2^4.
    * And 2^10 is roughly a thousand a good thing to remember.
    * So, this is 1024 * 1024 * 16.
    * And a thousand times a thousand is a million, times 16 is 16 million little dots.
  - So the accuracy with which you learn how much red is in a specific pixel in the image, is you get an accuracy of taking all possible values from 0 to 1, partitioning it to 16 million equally sized partition, and you pick which one of those intervals it lies. Still not perfect, but spectacularly accurate close approximation.
  - So the way that you are doing it is by, for each pixel, having 24 Rights or Lefts for each R, G, and B.
  - And if our frame has, say, 4096 pixels on X axis, and 2048 pixels on Y axis..
  - THEN that means that we need 2048 * 4096 * 24 * 3 (Y by X pixels, by 24 bits per channel, by 3 channels) bits to specify a frame.
  - A bit: there used to be a world of possibilities. We get told a bit, and that cuts that world of possibilities in half.
  - So above, we are going to be told 24 bits total to learn the total amount of contribution of red.

- So there are two levels of approximation:
  - FIRST: the discretization of space. We chopped up space into pieces. And we believe that if we chopped it up sufficiently fine that each pixel only has a single color in it.
  - We can think of the frame as a function (with 'zed' being the amount of red). We pick the domain of the function so that we have color constancy. We're approxmating a function (i.e. there isn't an average color, just one color per pixel)
  - SECOND level of approximation: we pick a color depth large enough that our eye can't tell the difference.
  - That is why Apple calls it "retina display" number of cells per frame, and number of bits is sufficiently large for color, that we can FOOL your eye. The amount of information it pushes out saturates your eyes ability to perceive information.
- So, if we do the 2048 * 4096 * 24 * 3 calculation, that number is going to be really, really big.
  - But many files on our computer will not be as large as this! Why is that?
  - that is because most files are *compressed.*
- So, some file formats for images might be RAW or PNG GIF or JPG. (left is uncompressed, right is compressed)
  - Now an aside, GIF as an "animated" thing is a relatively recent phenomena. Format itself has been around for 30 years.
  - uncompressed is easy. GIF and JPG, however, aren't just compressed, but what's known as "lossy compression"
  - If we want to save on space, we don't have to use all of those bits we were talking about above!
    - We could say "okay, look, instead of assuming an a priori fixed grid...
    - What I will do is determine rectangles inside the scene for which the color inside the rectangle is exactly the same
    - And we do that for the entire scene (rectangles don't need to all be the same size)
    - So, now, we assume, we first are told the partition of the image into those rectangles. And to learn this, all we need to know are the dots. If we learn all the dots, we can do the line fitting ourselves.
    - So, in our message, we get told a number, which will be no more than our number of pixels. it will be "Blah" number of dots (it must be less than number of pixels because the full number of pixels is our finest partition).
    - Then after knowing that, we get told where the dots are. Then we can do the partition ourselves (i.e. figure out where all of the rectangles are).
    - Then we either need a numbering scheme, OR we need to learn what the color of the rectangle is right away so that we don't need to come back to it.
      - To Recap The Above: So we need to know how many dots there will be. And we find that out via the same "bit" game.
      - By playing that game, we learn how many dots we have.
      - Next thing we need to know is the coordinates of the dots.
      - Then once we have that, we can start learning about the colors in each square
      - So the structure of the message will be: number of dots, followed by coordinates of dots, followed by colors of induced rectangles

      .
    - BUT, we need to agree on what is rectangle is 1, and which is number 2, etc. So you know that which amount of red goes into which rectangle.
    - So how do we do that numbering scheme?
      - So let's say we start in upper left hand corner. and we number things as 1, 2, 3, 4 from left to right.
      - So we are looking at the rectangles that touch the boundary from left to right. It is unambiguous.
      - So, interestingly enough, we can agree upon a scheme, such that even if all we are told is the coordinates of the dots, that is ENOUGH to define the partition. So we don't have to communicate any more information. So no matter what the squares might be, we would number them the same. No additional information is needed. We have what is known as a canonical representation. The method is sufficiently unambiguous that we would end up with the same number. And the method is always start at the top left and look at the top boundary. Top boundary broken into pieces by vertical lines.

        - So number is uniquely defined by the partition. Don't need to specify any more information.
        - And what we have achieved by this: potentially sending a lot less information than this.

- so a major consideration is how many blobs of continuous color we have.
  - If we have only one channel (say grayscale), we want our image to have many squares of constancy.
  - so the simplest thing is if your entire image is a single color all we need is one point (the lower right, as upper left is implicit) and a single color. Boom. All the information we need.
  - Not quite as good but also pretty good for compression under this scheme: a checkerboard
    * If the checkerboard is really big, then better than specifying all of the specific rectangles, you could instead specify the geometric pattern, i.e. a method of constructing the image in english.
    * But how do we know which of our two compression schemes we are doing?
    * EASY, it could just be the very first bit of the message.
    * If first bit is L, interpret as "english" description of geometric pattern, if first bit is R, then interpret it as the above bunch of rectangles.
    * So we might have a bunch of these different languages, two completely different ways of representing, and we can achieve which one by just a small number of bits.

- File Formats
  - so RAW and PNG are so called lossless formats
    * They try to send less information than the complete method, by employing schemes that are maybe a little more sophisticated, but similar in spirit, to what we described above. We identify areas where the value doesn't really change a lot.
    * But the bottom line is, the receiver gets the exactly same image as the sender.
  - On the other hand, GIF and JPG are LOSSY. And moreover, they come with a parameter
    * You won't get everything from the original.
    * But the algorithms for doing this are very sophisticated.
    * From a visual percetption view, not all bits are created the same.
    * so imagine a scene where there are mountains up front, and there are stars up above. We maybe don't really care about what is happening in the shadow down in the mountains, but it DOES matter to us that we see the starts up above. So if we are going to distort and lie and not represent things with enough accuracy, it is much better if it happens down in the shadows than up in the stars.
    * So the algorithms that do the encodings for these things are actually aware of this. So the person who created this with the raw information, has some understanding of visual perception system and takes that into account.
    * But the thing that takes the raw information does NOT understand what the mountain is or what a star is. it DOESN'T.
    * But what it DOES know, is that the star pixel is very, very different in intensity of color from its surroundings. So, for every pixel, it can tell that there are these huge spikes of white sticking out.
    * So the principle that the encoder operates can be largely though of as, there is an actual function sitting on top of that domain. How can we push some parts of it down, some parts of up, so that we can create large blobs of constancy in the function.
    * It's a little bit like a puzzle.
    * And in fact, it is even more complicated than this. And the reason why it is more complicated is actually pretty primitive. There are things like lines and spikes that are very important to survival.
    * if a line is going to attack us, we need to know what it is and respond
    * So our brain is very sensitive to certain patterns, and less sensitive to others
    * This is also known to the JPG compressor

- So we can imagine that we have a 1 Dimensional thing, that is a function that is some amount of red (or whatever color). And so that gets us a curve.
  - but then we can add in a new function, a smoother function, that is easier to transmit (that gives us a different curve).
  - And the amount of distortion is the AREA of the differences between the two curves. That is how much you distorted reality. So if you actually transmitted reality, there would be zero distortion.

  - As it turns out, this area is not a very good measure of distortion
  - And the reason why is because of the way that our brain works, certain KINDS of distortion is far more important than others. Could be that some parts of large area of distortion our brain CAN'T actually figure out. and some parts of small area of distortion are easily perceptible

- Why does this matter to computer science? This has changed THE ENTIRE STRUCTURE of the music industry.

Break

- Coming back after break, there is a "Picture of a hipster?" Guy with a mustache? AND a record player? instant hipster, right? (See pictures at end of notes).
- "Without music, life would be a mistake
  - This is a picture and quote of Friedrich Nietzche!
  - Teacher will give you an A if you take an exam on any of his major books.
  - "The Genealogy of Morality" you take a test, and then you get an A if you pass.
  - Teacher really likes Nietzsche.
  - Music really plays a big role in teacher's life.
  - It was 1980 something. Teacher was 11 or 12. And CD player was invented.
  - And across from his house in Athens, Greece, there was someone who worked at Philips, who invented CD players essentially.
  - and this was amazing transition. It was like life before CD and life after CD.
- So CD, if we remember right, is about 650MB (megabyte) The M/Mega is for million. Byte is for 8 bits. Which is the same as 8 "Left/Right symbols"
  - So a CD can be thought of as a long sequence of 650 million * 8 Left and Right. That's it!
  - we call sequences of 8 L/R symbols bytes.
  - The Mega is because we are talking about millions of bytes.
  - And 650 just happens to be the capacity of a CD.
  - with a CD, you can think of it as taking a long, spiral, spool. And taking that spool and creating holes in it (and either there is a hole or there isn't a hole)
  - And these things are evenly spaced.
  - So we can think of this sequence of bits. If we want to say "left" we leave it alone, if we want to say "right" we punch a hole.
  - So to read the cd, you shine a laser above it, and either it gets reflected because there is no hole, OR it does go through a hole. And there is a reader that says "I am getting light I am not getting light I am not getting light etc. etc."
  - And so then THIS is supposed to represent music.
  - and moreover, the interesting thing about it, is that it is supposed to represent music in a lossless way.
  - Let's be more precise about what we mean by this.
    * We draw a graph. And if we are quiet we are at 0.
    * Plot is location of the center of our ear drum. Time is the X axis, and what is being recorded is the distance from center of ear drum from it's resting position, where by resting we mean the absence of sound.
    * So inside ear there is a surface we will call that the ear drum. Let's pretend it is big and vertical.
    * As we communicate, ear drum moves in and out. We can capture the state of ear drum by capturing the location of its center in space.
    * second thing: absolute location in x,y space is irrelevant. What matters is it away from the resting position in this position or that position.
    * And that if you rotate your ear drum so that it is now horizontal.
    * function: input to function is time, output of function is the distance of the center of eardrum from resting position. And that "positive" means out, and "negative" means in.
    * Making assumption that all of our eardrums are the same.
    * Also observe that this is how a microphone works. What is microphone? An ear drum that records an electric signal that corresponds to the displacement. If you go to a concert, with a microphone, claim is that your eardrum and microphone would move together, would 'hear' the same thing.

    * But music, all music, is just this. it is a one dimensional function in time.
    * There is absolutely no difference between this function, and the intensity of red that we were looking at before.
- So, CD is the result of taking the result of taking the possible range of this function, dividing it into 2^16 possibilities, and literally saying where, in time, is our function.
- We'll pick this up next time so don't freak out.
- So CD, was the equivalent of those uncompressed formats of RAW and PNG.
- so CD just told you the intensity of each color in each pixel.
- And if you did that, the maximum amount of music you could fit was about 70 to 80 minutes. But this is under the assumption that you are using the 'stupid' encoding
- What killed music industry was MP3s, which managed to achieve MASSIVE compression, which is what allowed for transferring of music.
    * Why is MP3 compression capable of driving music down by a factor of 20 or 30
    * It is because it is super clever.
    * Remember how we could have areas of large compression that don't matter, and areas of small distortion that matter a lot.
    * There are some people that figured out how your brain processes sound. There are all sorts of things that you are basically not smart enough to hear. In the presence of certain sounds, other sounds become inaudible. They are there, and if someone tells you they are there, you'll hear them. But if you just play the music and something else is there and ask you if you heard it you would say no
    * So they figured out how to trick your brain.
    * So they introduce distortion that, to you, doesn't feel like distortion.
    * because they know that, based on what is before and after in the song, there might be certain parts of the song that you just can't even perceive. How you perceive music comes from what was played before and what was played after. If human brain isn't capable of capturing it, then it just throws it away!
    * DSL also played a part, and those came together at the same time
    * and now music industry is totally different.
    * And the true killer was someone's capacity to figure out what functional approximation has a perceptual component.

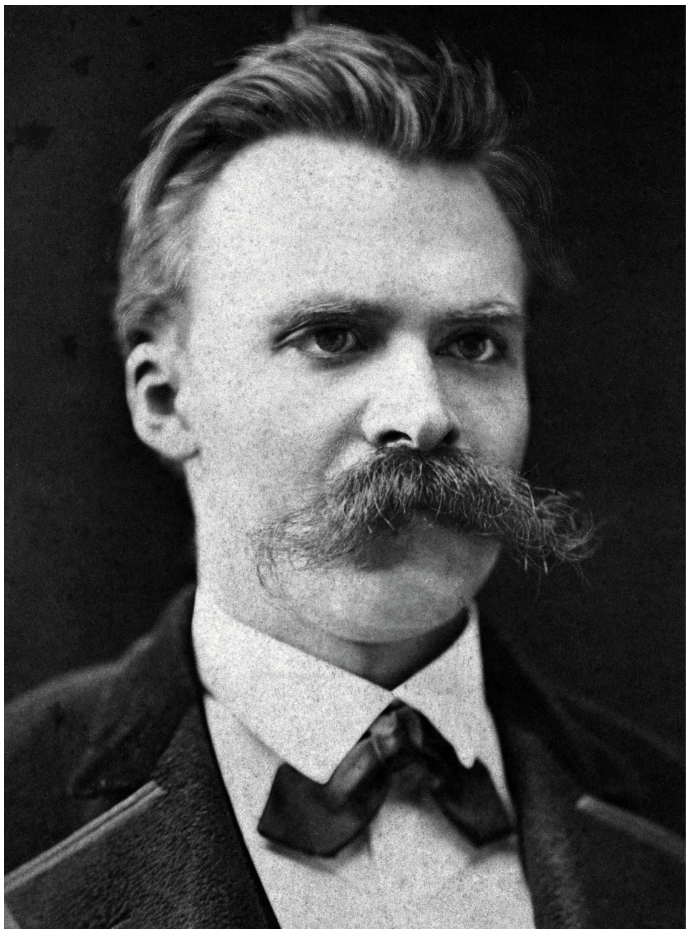

Figure 1: Nietzsche

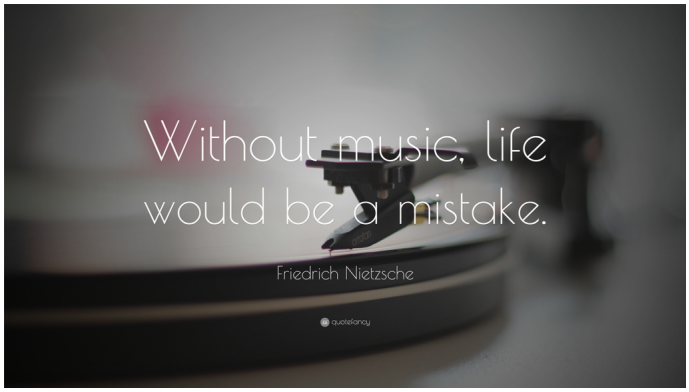

Figure 2: Nietzsche Quote