UNIT-4

Image Visualization

Contents

Image Data Representation

What is an image?

An image is a two-dimensional array, or matrix, of pixels. Pixels have square shapes, i.e., an aspect ratio of 1 to 1. Every image pixel contains one or several scalar values. Gray scale, or monochrome, images contain one scalar value per pixel. Color images usually contain three such scalar values, corresponding to a RGB or HSV (hue-saturation-value) encoding of the pixel color.

Image Data Representation

An image is more than a uniform two-dimensional dataset. Monochrome images contain one scalar attribute per pixel, indicating the luminance, or intensity, of each pixel. Color images contain one color attribute per pixel. In practice, color is represented as a triplet of scalar attributes, which correspond to a RGB or HSV color encoding. Value of a pixel is considered constant over the entire pixel surface. Images use a piecewise constant interpolation of luminance or color samples located at the pixel centers.

Image Data Representation-2D Image

First, use a full float resolution to encode each image data attribute, instead of the less-precise 24-bit or palette-based formats. Precision is essential to perform several operations on images without losing accuracy.

Second, allow images to store any number (and type) of data attributes. Besides luminance or color, image datasets can store vector or tensor data attributes. This flexibility allows to perform a wide range of processing operations on image data.

Finally, representing images as uniform datasets with floating-point attributes allows us to directly use many visualization algorithms on images without any modification

Image Data Representation – 3D Image

Image datasets and image-processing operations are not restricted to two dimensions. Two-dimensional (2D) images are still the most common, the vast majority of image data representations and imaging algorithms. Three-dimensional (3D) imaging is an indispensable tool in medical sciences, in the visualization and analysis of CT and MRI datasets, so that 3D imaging and data visualization have become tightly interconnected disciplines.

Image Processing and Visualization

That images can be represented as 2D scalar-attributed datasets- What is the place of image processing in the visualization pipeline?

The output of the complete pipeline is an image, which is typically displayed and analyzed to obtain the desired insight into the visualized data. To enhance this image for better understanding of the encoded information-contrast enhancement and color adjustment image processing techniques

Example: An application that produces a 2D uniform, scalar-attributed dataset at some stage of the filtering process a slice extracted from a medical dataset.

A color-coded height field or a two-dimensional flow texture representing a vector dataset.

To visualize such results is to map and render them using a suitable scalar to-color mapping. However, imaging operations can be applied, by performing image segmentation followed by shape analysis. In this scenario, the imaging pipeline is integrated as part of the visualization pipeline

Shape representation and Analysis

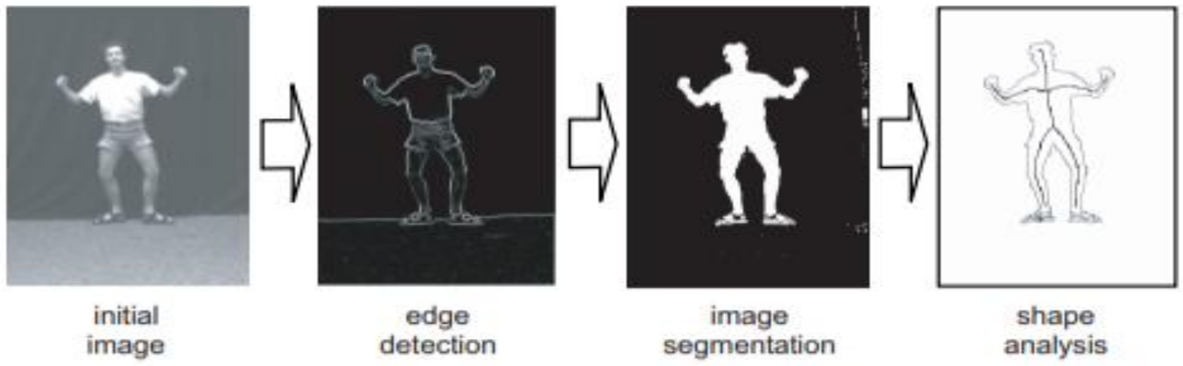


Figure Imaging and shape analysis pipeline.

Manipulate the information present in images at a higher level than pixels. Consider an image that contains a 2D slice from a 3D volumetric medical dataset, certain anatomical structures share the same scalar value, or values located within a specific scalar range. To extract such structures or shapes from image data using algorithms such as contouring and thresholding.

Limitations in this scenario:

First, local algorithms such as contouring and thresholding cannot detect and extract shapes that are based on complex, nonlocal definitions.

Second, some applications require shape descriptions that are richer than just the set of cells contained in an unstructured grid created by a selection operation.

Shape representation and analysis is a research field concerned with models and methods for representing, extracting, and analyzing shapes. The crossroads of digital imaging, perception, computer vision, and computer graphics. A shape is defined to be a usually compact subset of a given 2D image that globally exhibits some properties. Typical properties include the

Shape geometry (form, aspect ratio, roundness, or squareness),

Shape topology (genus, number, and ramification of the boundary protrusions),

Shape texture (color, luminance, shading).

Shapes are usually characterized by a boundary and an interior.

To identify the boundary:

All pixels located inside a shape has a value true and all outside as having a value of false those are boundary pixels. Boundary pixels are interior pixels that have at least one neighboring exterior pixel. Interior and exterior pixels are also called foreground and background pixels.Images having a boolean attribute value per pixel are also called binary images.

Binary images can be created from other representation types, such as closed polygonal meshes or implicit functions, by a process known as voxelization.

Voxelization assigns a foreground or background value to each voxel depending on whether The corresponding point is located inside, respectively outside the given input surface.

The process of the test is done in parallel using graphics hardware.

Shape analysis

First, images are acquired- scanning or photographic technology. Next low-level imaging operations are applied to remove noise and prepare the image for shape extraction.

In the shape extraction step, various properties of the shape (geometry, topology, texture) are used to detect and separate one or model shapes from the raw 2D image.
Finally, these shapes are analyzed to extract high-level, application-specific information
Basic Segmentation
First, the operations in shape analysis are the extraction of shapes from a given input image.
The first step in extracting such shapes is to segment or classify the image pixels into those belonging to the shapes of interest, also called foreground pixels, and the remainder, also called background pixels. Segmentation is strongly related to the operation of selection, which selects certain dataset points or cells based on their properties.Segmentation a dataset operation that creates a new boolean data attribute that has the value true for foreground pixels and false for background
Advanced segmentation: Normalized cuts: Another approach to image segmentation that partitions the input image into N segments, where N is a known value, is provided by normalized cuts. The image I is reduced to a graph G = (V,E). For each image pixel, we construct a graph vertex v V . Edges e = (u V,v V) E encode the likelihood that two pixels u, v are in the same edge segment, or the similarity of the two pixels. To model this, this, we set for each edge (u, v) a weight.

$$w(u,v) = e^{K\|\mathbf{F}(u)-\mathbf{F}(v)\|^2} * \begin{cases} e^{K\|\mathbf{X}(u)-\mathbf{X}(v)\|^2} & \text{if } \|\mathbf{X}(u)-\mathbf{X}(v)\| < r \\ 0 & \text{otherwise.} \end{cases}$$

Here, X represent the 2D positions of the pixels corresponding to the graph nodes. F are the image components of the pixels, such as intensity (for grayscale images) or RGB or HSV color components for color images. The threshold r typically set to a few pixels, indicates that only pixels close to each other are linked by edges. As pixels get further apart from each other, either in 2D space or in feature (intensity or color) space, their similarity decreases. After constructing the graph G, we aim to partition, or cut, it into parts which are highly similar internally but highly dissimilar with respect to each other. To measure the quality of a cut that partitions G into two parts A, B (so that A U B = G, A ∩ B =) that satisfy our similarity requirements outlined above, we use the normalized cut metric given by

$$Ncut(A,B) = \frac{\sum_{u\in A, v\in B} w(u,v)}{\sum_{u\in A, v\in V} w(u,v)} + \frac{\sum_{u\in A, v\in B} w(u,v)}{\sum_{u\in B, v\in V} w(u,v)}.$$

The numerator of the two terms expresses how similar are the two parts A and B. The denominators normalize the metric, thus preventing cuts which “chop off” small graph parts from larger parts to which they are highly similar. Finding the partition (A, B) that minimizes N cut given by Equation can be efficiently done by eigen analysis methods. Once such a cut is found, the graph (and thus image) is split into two parts. Next, the algorithm can be recursively applied on each resulting part until the resulting parts are either too small, a maximum user-prescribed value of N cut is reached, or a user-given number of segments N has been obtained.
Normalized cuts is an attractive method when we have less constraints on the number, shapes, and boundary smoothness of the resulting segments, and when the segment borders are clearly reflected by the pixel-feature-wise similarity distance. Also, the method is effective when we want to compute a hierarchical top-down segmentation of the image into progressively smaller segments, rather than a single-level segmentation.

Mean shift: Yet another image representation is proposed by the mean shift segmentation method. Given an image I of n pixels having d color components $c_1,...,c_d$ per pixel (x, y), we regard the image as a d + 2 dimensional point cloud $P = \{(x, y, c_1,...,c_d)_i\}_i$. Next, we estimate the density $\rho : R^{d+2} \rightarrow R_+$ of P as

$$\rho(\mathbf{x} \in \mathbb{R}^{d+2}) = \sum_{i=1}^{n} k\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right).$$

That is, we convolve the point positions $\mathbf{x}_i$ with a parabolic radial kernel

$$k(x) = \begin{cases} 1 - x & \text{if } 0 \leq x \leq 1 \\ 0 & x > 1. \end{cases}$$

Having the density ρ, we next iteratively shift the point's x_i in the direction of the gradient ∇ρ, and recomputed the density ρ following Equation 9.18. Repeating the process for several tens of iterations effectively makes the point's xi converge into their local density maxima. Finally, we cluster the final point set delivered by the mean shift procedure by grouping all points xi that are closer to each other than a user-prescribed small distance. Each point cluster will represent a segment of our input image I. Since the identities of the points do not change during the mean shift, we can trace back from points in a cluster to their original positions in 2D, and retrieve the spatial cluster extent. The cluster value in color space gives us the average segment color, if desired. Mean shift can be efficiently implemented for low dimensional spaces ($d + 2 \leq 3$, e.g., for segmenting gray scale 2D images), by evaluating both its most expensive component, the density estimation and the relatively cheaper mean shift itself, in parallel using GPU programming.
Image foresting transform: Another segmentation method that treats the input image as a graph is the image foresting transform (IFT) method. The method starts by constructing a graph G = (V,E) where each image pixel is a node, and pixel neighborhood relations create edges. The edges (u, v) ∈ E carry weights w(u, v) which encode the Euclidean distance between their colors. Next, assume we can compute, for any path π ⊂ G, a positive cost f(π). We define an optimal path ending at a node v ∈ V as being the minimal-cost path from all possible paths over G that end at v. The IFT then computes an optimal path forest, or set of disjoint trees, that covers all nodes in G, so that each path in this forest is optimal in the above sense. Intuitively, the IFT partitions G into several disjoint subsets. Each subset has a root node (the root of each forest's trees) so that all nodes in that subset are closer to the root than to any other root.
The above general IFT framework can be effectively used for image segmentation as follows. First, we select a few pixels x_i ∈ I in the input image I, which we label by distinct-valued labels L(i) = i. All other pixels are labeled with a value L = 0. Next, for all trivial paths consisting of a single pixel x we use as cost function

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if } L(\mathbf{x}) \neq 0 \\ \infty & \text{if } L(\mathbf{x}) = 0. \end{cases}$$

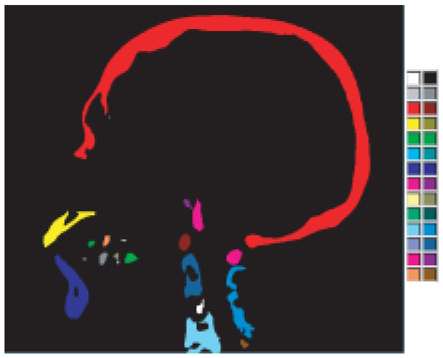
The cost of a path $\pi = (v_1,..., v_n)$ to which we concatenate a node x to yield the path $\pi = (v_1,..., v_n, x)$ is next defined as $f(\pi) = \max (f(\pi), w\{v_n, x))$. Running the IFT on an image having a few labeled pixels will partition the image into disjoint segments whose pixels are similar with the labeled pixels.

Connected components:

Connected-component detection proceeds as follows:

First, a segmentation operation separates the image into foreground and background pixels. Let us, for simplicity, assume that the segmentation creates an integer attribute comp for the image dataset, where the background and foreground pixels are marked by the reserved values BACKGROUND and FOREGROUND, respectively. To hold the identifier of the next component to be found, we use an integer variable C, which we initialize to zero.

Next, we scan the image, e.g., row by row. When we find the first FOREGROUND pixel, we set its comp value and that of all its direct and indirect FOREGROUND neighbors to C, increment C, and repeat the scanning until all FOREGROUND pixels are exhausted.



At the core of the algorithm is the process of finding all FOREGROUND pixels that are direct or indirect neighbors to a given pixel. This operation, also known as flood fill in computer graphics, can be easily implemented using a stack data structure holding pixel coordinates. The code of the connected components algorithm is sketched.

The floodFill function assumes pixels are connected, i.e., part of the same component, if they have the FOREGROUND value and are vertical or horizontal neighbors of a pixel already in the considered component. This choice would not identify diagonally neighboring pixels as being connected, and would place them in different components.

If we want components to be diagonally connected as well, we have to add four more push statements to the floodFill function, corresponding to the diagonal neighbors $i-1, j-1$; $i-1, j+1$; $i+1, j-1$; and $i+1, j+1$ of the current pixel i, j.

It is also noteworthy to add that several speed and space optimizations can be applied to that have been omitted here for conciseness.

After the function connected Components returns, the array comp contains every pixel marked by the nonnegative identifier of its connected component or the BACKGROUND value.

Each component is color-coded by a different hue. We can now easily distinguish the largest component, marked in red, which corresponds to the cranial bone.

If desired, we can now filter the components based on size. For this, we count the number of pixels in every connected component, e.g., using a scan-line traversal of the image or a flood fill from the first pixel belonging to that component, and next mark components whose size is below the desired threshold by the BACKGROUND value. Removing small connected components is also known as island removal.