Abstract

In this chapter we continue our discussion of "multi-aspect" modular design initiated in Chapter 4. Here we will outline how this design pattern may specifically be applied to modules dealing with image annotations, and we will break down the suite of geometric, dimensional, and image-metadata considerations that should be addressed by image-annotation modules. We will also begin analyzing questions related to how image-annotations related to the morphology and geometry of image regions (their area, convex hull, bounding rectangles, overall color properties, and so forth) can be integrated with notations for more complex image feature-vectors associated with radiomics (various kinds of texture analyses and feature-extraction techniques). Our goal is to form an account of the representational capabilities prerequisite of a general-purpose bio-image annotation system which would cover the broad range of image-based investigative modalities in biomedical research and clinical/diagnostic practice, from radiology and microscopy to radiomics and feature-based predictive analytics.

Chapter 7: Multi-Aspect Modules and Image Annotation

1 Introduction

Modularity is a canonical principle in software engineering: it is almost universally accepted that modularity is desirable, even if not practiced as ubiquitously as that would imply. In this chapter we will not attempt to offer any new insights into modular design in general, but will instead focus on the version of this paradigm which we introduced last chapter as "multi-aspect" modular design. The key detail about this approach is that each module, in the general case, provides features associated with a range of software-development aspects, such as data persistence, GUI design, serialization, and capabilities related to runtime-reflection and participating in multi-component workflows. Multi-aspect modules provide a compromise between monolithic stand-alone applications and more narrowly focused modules typical of conventional modular design. Our discussion of modules' "aspects" builds off of ideas presented in earlier chapters, and the intuitive "Semiotic Saltire" schematic diagram, where database, serialization, GUI (or "visual objects") and "code models" were pictured as four aspects circling around a central data type (mimicking the "saltire" pattern of, e.g., the Scottish flag).

1.1 Comments on Procedural and Database Aspects

For our purposes, we will define a *multi-aspect module* (which we can just call "modules" when discussing only this specific form of modularity) as a *code library* that, in typical cases, includes procedures addressing multiple *aspects* of software implementation. In particular, modules will have groups of procedures concerning data persistence/database integration, serialization, and GUI visualization. In addition, modules will typically have some notion of a "code model" and have capabilities for dynamically invoking procedures or functionality exposed by the module. For sake of discussion, we will refer to modules' "code modeling" dimensions in this sense as "procedural exposure."

Before addressing image-annotations in particular, we will make some general comments about the four "aspects" we have focused on, namely serialization, GUIs, data persistence, and (what we are calling) procedural exposure.

The procedural-exposure aspect includes (at least) two distinct possibilities. First there is the goal of *runtime-reflection*, or calling a modules' procedures by describing the desired procedure and its parameters. In this scenario, the origin of requests to execute reflected procedures lie outside the module's own code. Moreover, these requests are not compiled in to some application or some other module, but rather dynamically are dynamically generated; as a result, the relevant module in the procedure cannot be called directly, but rather a request to run that procedure must be encoded indirectly. If a module is implemented as a C++ code library, for example, then it is straightforward to implement procedures as externally accessible to other C++ modules, which can call the relevant procedure as an ordinary C++ function. Choosing procedures to expose to sibling modules in the same language and environment as the original module is indeed one dimension of code modeling, but runtime reflection involves a different scenario, where the procedure requests originates from a source which is *not* the same language as the original module, or where for some reason the procedure cannot be called directly. In these cases, the caller encodes a data structure describing the procedure and its parameters, and the module must supply capabilities to interpret such requests and route them to the correct procedures.

The second manifestation of *procedural exposure* is similar to runtime reflection, but in this case requests need to be mapped to specific procedures implemented within a module. Instead, modules may define different "services," "endpoints," or "request handlers" — following our precendent in Chapter 6, we will use the term *meta-procedures* — which are functionally similar to procedures (in the sense of having input and output channels and being executed as a self-contained process) but may not be directly implemented in a single function-body. Request handlers are characteristic of web services: web URLs often encode the name of an endpoint which is available through the corresponding web application, but the names of actual procedures defined in the computer code responsible for the web application is rarely exposed to the World Wide Web. A *meta-procedure* in this sense is similar to a service or API endpoint provided by a web application, except meta-procedures do not necessarily communicate or encode data according to web protocols (such as HTTP).

Some clarification may also be in order with respect to aspects involving data persistence. We assume that modules provide code which cuts across multiple aspects in a relatively self-contained manner; for instance, each module provides its own serialization and GUI code. This approach differs from conventional modular design, where more likely a single module will address only a single "aspect" — for instance, a component dedicated to data analytics would be less likely, according to convention, to include serialization and GUI classes. Our notion of "multi-aspect" modules therefore assigns greater scope and autonomy to modules, almost as if each module is a kind of mini-application. However, modules are still intended to be pieced together with other modules to form applications proper. Multi-application modules are not intended to be *entirely* self-contained.

In the area of data persistence, it would be impractical for every module within an application to maintain its own separate database. We assume therefore that applications are

designed with data-persistence capabilities that are independent of modules which are part of the application. Nevertheless, the details of how data is encoded for the purpose of data-persistence can be deferred to individual modules in turn. Each module, according to this model, can then autonomously provide data structures encoding how all the data managed by the module, at a specific point in time, should be encoded. The modules do not communicate with databases directly, but they are responsible for providing raw data which the database eventually receives.

One goal of this setup is to delegate control of how persistent data is structured to modules, while abstracting low-level database details away from individual modules' responsibility. As a result, modules should be designed around generic database capabilities, rather than any specific database model. Taken to its logical conclusion, this idea implies that modules need not commit to any particular database *architecture*, such as SQL or different flavors of NoSQL. This degree of genericity is made possible by practices such as "pre-persistent" representations, discussed in Chapter 6. According to that model, applications can set up a generic data-representation format which is agnostic to specific database architectures, and then receive data from individual modules within that format. The procedures responsible for communicating with databases directly, then, would translate the "pre-persistence" representations into information organized according to the rules of the specific database being targeted.

This discussion remains somewhat ephemeral without going into further details about the nature of these "generic" representations, which we touched on in Chapter 6 (see for example Figure 4 in that chapter). At this point we will note only that generic pre-persistence models can employ a diversity of architectural strategies at the *representational* level, whether or not they map directly onto low-level database structures. For example, pre-persistence representations can employ key-value "properties" annotated on multiple kinds of sites within data structures, consistent with property-graph models; or employ hypergraph aggregation structures; or encode SQL-style tables with fixed column layouts depending on record kinds (each record-kind being akin to a table, but without implying that records must be grouped together by kind into isolated tables rather than potentially forming other sorts of collections). Such pre-persistence structures may project onto concrete database features in some scenarios, but there are strategies to encode structures endemic to one database architecture within the terms of alternative architectures, even if the results do not have the same level as optimization.

In Chapter 6 we discussed virtual machines for query-evaluation, which are also applicable for working with pre-persistence representations in a multi-aspect modular context. Virtual Machines can be engineered to operate directly with data structures formulated according to pre-persistent representations. The set of operations exposed by VMs in this context could likewise include features for marshaling information structured according to "pre-persistence" representations into formats endemic to specific database genres.

Having therefore clarified certain details specific to data persistence and "procedural aspects" of modules in general, we can present specific examples through the case of image-annotations, continuing the outline of annotation-related data formulated in earlier chapters.

1.2 Assessing the Proper Scope of an Image-Annotation Module

At the end of Chapter 4 we suggested limitations in (informal) "application-centric" workflows, where usage-patterns involving specific popular applications tend to become entrenched and relatively inflexible. We argued that these usage-patterns demonstrate a kind of inertia which inhibits the emergence of more flexible workflows where components can be pieced together in a more open-ended fashion. Workflows are more flexible when their component parts are more narrowly focused, such as code libraries, rather than monolithic applications. This is (at least in part) due to narrower components being explicitly designed with the anticipation of interoperating with other components in order to be useful, whereas standalone applications are built to serve users' needs on their own; interoperability with other components may be desired capabilities provided through plugins, extension, or "advanced" features, but they are less crucial to the applications' usability in the first place.

Technical discussions of software workflows accordingly tend to focus on components which are designed to be used as one element within a larger computing environment. Similarly, modular design emphasizes the virtues of software systems being assembled from operationally isolated parts. Both workflow designs and modular software-development, then, are grounded in relatively streamlined components targeted at specific areas of functionality. However, defining the scope of components' desired features too restrictively leads to other sorts of complications, which we will discuss in this chapter. There is therefore a tension between conceptualizing workflows and modular design too narrowly or too broadly: components which act like monolithic applications present the risk of "ecosystem fragmentation" and the failure to achieve modular design spaces which are flexible and open-ended with respect to how components are pieced together, but components which are too narrowly targeted to individual software-development concerns can require extra effort to be integrated into overarching projects, mitigating the potential benefits of modular design. Multi-aspect design attempts to engineer an optimal balance between these two extremes.

This chapter will take image-annotations as a use-case motivating the basic ideas of Multi-Aspect Modular design. This chapter will also continue our analysis of image-annotation data, which we reviewed at the end of Chapter 6

in the context of the Annotation and Image Markup (AIM) format. Image annotation presents a good case study for data-integration problems in general, because properly interpreting annotation data requires integrating at least four different data models: annotation (geometric) descriptions themselves; visual presentation details (how annotations should be displayed on-screen); image-acquisition metadata (recording the provenance of image series, color depth, resolution scale, and other factors which determine how images' pixel data should be construed optically) and clinical/diagnostic background which permit image-annotations to be employed as a form of biomarker. Because image-annotations are connected to each of these four separate domains (and possibly others, depending on how one wishes to demarcate domain-boundaries), bioimage-annotations are likely to play a role in many different diagnostic, research, and/or clinical-decision contexts, meaning that annotations will in general be synthesized with different forms of associated data (such as tissue biopsies, treatment plans, health records, and so forth). Annotations therefore offer a convenient window on the sorts of issues that arise when data structures with significantly different profiles need to be merged into a single information space (for purpose of analysis, Machine Learning, unified data persistence, interoperating GUI components, and so forth).

Image annotation and segmentation is an important analytic process in many scientific, technical, and commercial fields. Nonetheless, there are few standard formats for describing and representing image annotations, and those which do exist tend to be used in specific, relatively narrow contexts.[1] This is not a new observation; Daniel L. Rubin *et al.*, in 2007, note that:

> Images contain implicit knowledge about anatomy and abnormal structure that is deduced by the viewer of the pixel data, but this knowledge is generally not recorded in a structured manner nor directly linked to the image. [Moreover,] the *terminology* and *syntax* for describing images and what they contain varies, with no widely-adopted standards, resulting in limited interoperability. The contents of medical images are most frequently described and stored in free-text in an unstructured manner, limiting the ability of computers to analyze and access this information. There are no standard terminologies specifically for describing medical image contents — the imaging observations, the anatomy, and the pathology. [N]o comprehensive standard appropriate to medical imaging has yet been developed. A final challenge for medical imaging is that the particular information one wants to describe and annotate in medical images depends on the *context* — different types of images can be obtained for different purposes, and the types of annotations that should be created (the "annotation requirements" for images) depends on that context. For example, in images of the abdomen of a cancer patient (the context is "cancer" and "abdominal region"), we would want annotations to describe the liver (an organ in the abdominal region), and if there is a cancer in the liver, then there should be a description of the margins of the cancer (the appearance of the cancer on the image). [17, pages 1-2]

These challenges inspired the Annotation and Image Markup project, which "provides a solution to the ... imaging challenges [of]: No agreed upon syntax for annotation and markup; No agreed upon semantics to describe annotations; No standard format ... for annotations and markup."[2] However, AIM has been adopted most noticeably in cancer research and radiomics, less so in other biomedical areas.

One obstacle to formalizing image-annotation data is that annotations have a kind of intermediate status, neither intrinsic parts of an image nor merely visual cues supporting the presentation of the image within image-viewing software. Many applications exist which allow markup or comments to be introduced with respect to an image. From the application's point of view, these annotations are part of the application display, not part of the image — analogous to editing comments that might be added to a text document by a word processor or PDF viewer, which are records of user actions, not intrinsic to the document itself. Indeed, one mechanism for recording image annotations in DICOM (the "Digital Imaging and Communications in Medicine" format) is via "presentation state." The presentation state includes all details about how the image currently appears to DICOM workstation users, such as Radiologists, which could include markings they have made to indicate diagnostically significant image regions or features. Insofar as image-annotations are considered to be artifacts of image-viewing software, rather than significant data structures in their own right, there is less motivation for imaging applications to support canonical annotation standards.

Nevertheless, in many scientific and technical areas image annotations *are* significant; they are intrinsic to the scientific value of a given image as an object of research or observation. Image regions, segments, and features have a semantic meaning outside the contexts of the applications that are used to view the corresponding images, which is why it is important to develop cross-application standards for describing and affixing data to image annotations.

It is also important for image-annotation models to be broadly applicable and multi-disciplinary. While image analysis serves different goals in different contexts (e.g., segmentation of microscope images to detect cancer cells serves different ends than segmentation of street-level camera snapshots to study traffic patterns), there is always a possibility of analytic techniques developed in one subject area to be

[1] Current formats include AIM (Annotation and Image Markup), CVAT XML (CVAT is the Computer Vision Annotation Tool), DICOM-SR (Digital Imaging and Communications in Medicine Structured Reporting), PASCAL VOC XML (Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes), and COCO JSON (Common Objects in Context).

[2] https://wiki.nci.nih.gov/display/AIM/Annotation+and+Image+Markup+-+AIM

applicable for other image-processing problems, even if the practical outcomes desired of the analyses are very different. Furthermore, certain computational domains are similar enough to image analysis to warrant inclusion in a general-purpose image-annotation framework, even if the underlying data does not contain "images" in the conventional sense (not, for instance, captured via photographs or microscopy). For example, PDF document views, Flow Cytometry (FCM) data plots, and geospatial maps subject to Geographic Information Systems (GIS) annotations may all be considered images — by virtue of a semantic significance attributed to color and to geometric primitives as a way of characterizing phenomena observed or modeled through their data — even though such resources are not acquired by ordinary "image-producing" devices.

Most practical applications do not attempt to define "images" as such, and therefore to delineate the scope of image annotation overall.[3] Here, we consider imaging to be more general than just graphics obtained by a direct recording of the optics of some physical scene via cameras, microscopes, or telescopes. That is to say, the image acquisition process is not necessarily one where data is generated by an instrument which produces a digital artifact by absorbing light, so that geometric and chromatic properties of the image are wholly due to the functioning of the acquisition device. How broadly we do defining imaging *per se* — which remains an open question — affects the range of domains whose semantics could reasonably be incorporated into annotation frameworks. For example, if immunofluorescent Flow Cytometry (FCM) data plots are classified as images, then the numerical properties of the "channel" axis, with notions of "decades" and a "log/linear" distinction, become relevant to the annotation vocabulary for representing spatial dimensions and magnitudes.

After devoting the first part of this chapter to a relatively detailed examination of image-annotation data, the remainder of the chapter will turn attention to other bioinformatic categories. Here we will consider data-integration strategies in the context of modeling image-annotations (and image biomarkers) alongside other varieties of biomedical information.

2 Image Annotations: Core Data Models

The first step when formulating a general-purpose image-annotation data model is to consider the underlying representation of image "points" or "locations" in the first place. As we will show later in the context of AIM, this is a more intricate problem than it may appear at first glance.

An image annotation has the distinguishing characteristic of being a *geometric* object, but one whose meaning is only available *in conjunction with* an accompanying image (or 3D model, and so on for higher dimensions; we assume that the scope of image-annotations can be extended to include time points and intervals, and so be applied to 4D media as well). Any geometric entity, such as points, lines, or regions, therefore needs to be defined in relation to the accompanying image (sometimes called the *ground* image). For example, any magnitude in the annotation data needs to be evaluated relative to the size and resolution of the image. An intrinsic feature of any annotation-description is therefore the *scale of resolution* at which it applies to the ground image.

We might assume by default that annotations are always markings targeted at the ground image in its "internal" (100%) zoom level. In this case, it should be guaranteed that data about ground-image dimensions can be ascertained from whatever data structure or object represents the ground-image itself. For example, if annotation magnitudes are expressed in millimeters, the numbers are meaningful only after establishing the width and depth of the ground image in millimeters accordingly (assuming it is viewed at its "natural" scale, without zooming either in or out as an artifact of the visual display). Such image details must therefore either be an internal field or group of fields within the annotation data or else must be accessible through the object representing or referring to the ground image (here we will use the generic term "object" to mean an integrated data structure, not necessarily endowed with Object-Oriented designs). All annotations accordingly have an *intrinsic* scale which represents the *internal* details of how magnitudes in the image relate to points/intervals/locations in the ground image. This intrinsic scale may be different from the visible scale through which annotations are presented: if the ground image is zoomed in or out, any displayed annotations would need to be scaled equivalently. The particular dimensions of an annotation *insofar as they are viewed* in a software window, therefore, are artifacts of the *rendering* of annotations rather than manifestations of data which is *part* of the annotation.

A complicating factor, however, is how annotations are created in the first place. Suppose an annotation is marked on an image which the radiologist (say) views at 200% scale: the annotations visible to the radiologist are therefore double the size of their "intrinsic" data. Presumably any annotating software is capable of scaling the on-screen rendering to compensate for zoom effects. The *visible* rendering of the annotation is not the same thing as the annotation itself; so when a radiologists marks a one-centimeter length on a 200% zoomed image, the software would internally record the annotation data as scaled so that the corresponding length is one half-centimeter. All this is straightforward. However, in order to precisely record the conditions under which an annotation is created — perhaps so as to re-evaluate diagnostic findings supported by the annotation — it may be appropriate to notate the fact that *from the perspective of the user* who created the annotation in the first place, they

[3] The "Pantheon" project, characterized as a "platform dedicated to knowledge engineering for the development of image processing applications" (see https://hal.archives-ouvertes.fr/hal-00260065/document), offers one of the few attempts in imaging literature to rigorously define "imaging" and "image processing" in the first place. Pantheon (via its **Pandore** component) also includes an image processing *objectives* ontology.

were viewing the annotation (and likewise the ground image) at double-scale. For each annotation, then, one *may* wish to record the resolution at which it was viewed when first created, if this differs from the default ground-image scale.

Similarly, when an annotation is subsequently viewed by any user, the software will typically support zoom operations, so the *state* of the annotation currently visible may be different than its "intrinsic" dimensions. It should be possible to bundle the annotation *together with the view state* as a larger data structure associated with any annotation *and also* to isolate the annotation from any particular context where it is viewed. View-state details (such as zoom factors) are not significant to how the annotation is interpreted or analyzed, and as such should not be confused with *intrinsic* data for purposes of data-sharing. On the other hand, in some contexts it *is* desired to share view-state: consider a telepathology scenario where a doctor and a radiologist discuss an image from two different locations (in the sense of, say, two different cities). Presumably they should be able to synchronize their views so that they see the image and annotations at the same scale, with the same coloration, and so forth.

Considerations of view state also apply to details such as colors and line width. Ordinarily, colors are a presentation detail rather than intrinsic to annotation data; there is no geometric significance attached to a line being drawn as yellow rather than red, for instance. Colors may be pressed into service as a classification device: one may want to distinguish annotations playing different interpretive roles. For example, circles or polylines outlining a Region of Interest might be classified as playing a different role than line segments whose purpose is to calculate some biologically significant length. In this sort of situation annotating software may use colors as visual cues to the corresponding role. However, only the roles themselves, not the colors that signify them, are "intrinsic" to the annotation. Colors *are* internal to view-state data, and should be modeled within this data, alongside details such as image/annotation zoom factors. Similar points apply to opacity (if annotations are semi-transparent to allow some of the ground image to remain visible) and to line-width (lines may be thickened to make them more visible, but most annotation data structures consider lines to be hypothetically infinitely-thin extants which server merely to connect two points, establish a length, or form part of a polyline encosing a region). Similarly, line-styles (using dashed, dotted, or "wavy" lines, say, instead of straight ones) could be employed as a way of connoting roles or other non-geometric information — consider a system that renders annotations marked with less confidence via dashes rather than solid lines.

With that said, however, it would be premature to rule out the possibility that certain kinds of annotations would make *intrinsic* use of features such as colors, line-styles, and opacity. As a result, a general-purpose annotation data model should identify information which is *always* intrinsic to the annotation itself (such as geometric vertices) as well as view-state details which *usually* are presentational rather than intrinsic data; but should allow for some typically-presentational data, in specific contexts, to be treated instead as intrinsic to the annotations.

2.1 Magnitudes and Coordinates

Assuming, then, that we have accounted for ground-image details such as the image's intrinsic dimensions, and identified which annotation data is intrinsic and which is presentational, the annotation itself will be embodied via geometric entities (such as points, lines, and regions) referring to locations and intervals in the ground image. Typically an annotation is a geometric structure whose meaning is depending on some separate interpretive declaration — a point or line has no significance other than its geometric role in fixing the annotation's shape.[4]

As purely geometric entities, points thereby represent "locations" within the image, and line-segments represent intervals within image's internal "space." This language is inexact because it is subject to different interpretations. We could see image "locations" as individual *pixels*, which among other consequences has the effect that the building-blocks of locations are integers (there is no such thing, at least speaking literally, as e.g. a "half-pixel"). Alternatively, locations can be treated as abstract (dimensionless) points in the "space" inexactly represented by an image. This choice conforms to the general distinction between *raster* and *vector* graphics, where the former is pixel-based and the latter is more mathematical. Raster and vector have distinct use-cases, so a general annotation model should support representations conducive to both kinds of graphics

One consequence of this generality is to preclude *a priori* judgments about how magnitudes and coordinates should be represented. Image locations may be understood as integers in some contexts and as floating-point values in others. Moreover, floating-point values might have different degrees of precision in different contexts (including "infinite-precision," which allows quasi-real numbers to extend across as much computer memory as needed to achieve arbitrary degrees of precision).

It is possible for an annotation system to allow two different numeric types insofar as numbers designating locations have a different purpose than those marking quantities such as rotation angles or ratios/eccentricities; we might use integers or **float**s for point-locations but **double**s for magnitudes that are not fixed to spatial points. Note also that "quasi-reals," by which we mean numbers intended to approximate the full real-valued number line, do not necessarily

[4] If extra-geometric data such as colors are (in some context) treated as *intrinsic*, then geometric elements in the annotation play the additional role of providing sites for the corresponding extra-geometric indicators. For example, if line-color is intrinsically significant, than a line becomes an object which can be assigned a color, and so it plays the role of permitting the color-data to be expressed in the annotation, alongside its role in defining a shape. For the current discussion, however, we will not consider extra-geometric data along these lines; hence points and lines have no "meaning" within the annotation other than to construct a shape-representation which is interpreted relative to the ground image.

need to be encoded according to the familiar floating-point standards. There are alternative number systems, such as John Gustafson's "posits," which offer an alternative that may be better-performing than normal floats in some contexts [12]. All told, then, there are a variety of different number systems which could potentially supply magnitudes for annotation geometry.

As we will discuss below, number-system flexibility is typified by libraries such as CGAL (for computational geometry) more than frameworks targeted more specifically at annotations proper, including AIM, which only recognizes double-precision floating-point magnitudes. In CGAL, any assertion of magnitude is always dependent on the specific number system used to mathematically ground the geometric system in the first place. For example, annotation points representing image-locations by encoding distances from the image's left and bottom sides (or left and top, and so forth) are dependent on fixing a convention for whether numbers are expressed as integers, ratios, quasi-reals, or instances of some other number system. Therefore, another data point which is *intrinsic* to annotations are specifications of what kind of numbers the annotation uses in the first place (integers, doubles, single-precision floats, width/hight percentages vis-à-vis the ground image, etc.)

As we also mentioned in the context of CGAL, polar or "heterogeneous" coordinates (instead of familiar Cartesian coordinates) are utilized in some contexts. Additional special-purpose coordinate systems may come into play for higher-dimensional media; annotating 4D dynamic graphics, for example, could potentially be facilitated via Quaternions (see [2] or [15], for example).

In short, a general-purpose annotation framework should allow sets of annotations to vary over both the nature of magnitudes (integers, floats, etc.) and the coordinate systems to which those magnitudes apply. Any description of a collection of interrelated annotations should therefore notate such numerical details as intrinsic qualities of the annotation-set, even if not of particular annotations. In general, coordinate details would be fixed for multiple annotations in a set and therefore not re-declared for each annotation.[5] This implies that there must be some separate data structure representing an annotation "set" with higher cardinality than each annotation singularly. Depending on context, the extent of such a set may be all annotations on a given image, or image series (in the clinical sense of multiple images acquired at the same time as part of one diagnostic procedure), or larger image-collections in a database, and so forth.

This discussion points to several questions: how should we demarcate the extent of an annotation-set when defining a data structure which would hold information that applies to all of the annotations, such as magnitude and coordinate stipulations? Should the annotation-specific data model allow *nested* annotation-sets, or labeling annotation-sets with different roles (single-image, image-series, etc.)? Should coordinate information be unequivocably fixed across an annotation-set, or should individual annotations vary some of this data, e.g., some annotations expressed in Cartesian coordinates and others in polar? One possibility is to divide annotation-sets into subsets wherein every annotation shares the same coordinate setup, so at this *subset* level all coordinate data is homogenized.

An entirely separate issue is fixing units of measurement for points, lengths, and locations, such as centimeters or inches. Units may also be designated taking pixels as minimal spatial units, so that orthogonal distances are treated as (integer) multiples of pixels' width and height — we can speak of a horizontal line segment being 12 pixels long, for instance. Quasi-real lengths such as line segments (not perpendicular to the image sides) can be modeled as the hypotenuse of the corresponding right triangles whose shorter sides are orthogonal pixel-lengths, so that pixel-based units can be generalized to non-integer values. Scalable Vector Graphics (SVG) recognize 9 different length units (including percentage relative to the whole image). It is reasonable to assume that each of these units might potentially be preferred as the basis of annotations in different contexts (for reference, [18], [1], [11], [5] present useful overviews/extensions for SVG).

To fully specify an image location, then, we need to determine three different factors: the nature of the raw magnitudes involves (e.g., integers or floating-point); the coordinate system wherein points should be oriented (e.g., Cartesian coordinates, with axes related to the center or sides of the ground image, or polar coordinates); and the scale-units for lengths. Coordinate-system involves orientation as well as choice of a particular mathematical system; for instance, some graphics environments would center Cartesian axes on the center of an image, others on the top-left corner so that increasing numbers correspond to movement down or leftward; others the bottom-left or bottom-right, and so forth.

In most cases, all of this foundational data will be declared for groups of annotations rather than individual annotations. Within the scope of a single annotation, we can take for granted that a pair of numbers (say) unambiguously designates a point in the ground-image. However, this assumption is dependent on all the requisite background data being shared alongside annotation-specific data in any data-sharing scenario.

Assuming we thereby have enough infrastructure to rigorously designate individual points, annotation-shapes can then be built up as point-sets — subject to multiple interpretations. Point-sets may be used to described curved objects as well as straight line segments (and collections thereof) but for now consider just straight-line cases.

Assume that annotations' shape data contains two or more points that describe an open polygonal arc or a closed poly-

[5] Although one can envision scenarios where different coordinate systems would be useful for different annotations even in the same set; for example, mixing Cartesian and Polar coordinates for different regions of a single image, or expressing some locations via percentages of the image's overall dimensions (rather than units such as inches or millimeters).

line. There are several details to be considered. One is orientation: it is reasonable to model polylines, at least those which span a convex region, as point-sets where points are ordered in either a clockwise or counter-clockwise direction. Should the data model restrict how points or ordered, so that points are automatically sorted within the annotations' point-set or, alternatively, constructions which would yield inconsistent ordering be rejected?

This question amounts to the following: suppose we have a point-set which can be shown geometrically to span a convex region when the points are arranged in a certain order. Should that ordering be taken to be canonical, such that a different ordering declared among the points is interpreted as an artifact in how the points are assembled, which can be corrected when finalizing the annotation data? Or should annotations allow for self-intersecting polylines?

In the former case, should each annotation be given the option of clockwise or counter-clockwise ordering (insofar as chiral orientation is significant in some contexts) or should the points automatically be repositioned in clockwise manner, say? Moreover, how should these specifications be enforced? Should a clockwise-ordered point-set be considered identical to a second set with the same points but a different order? Or should self-intersecting point sets (assuming one simply creates line segments by following the points in the order given) be considered malformed? Also, how do we deal with anomalies such as multiple points being duplicated in the point-set? Should the extraneous points be eliminated, or should data with those characteristics be deemed malformed and unusable?

Another line of questions concerns closed polylines versus open polyarcs. How should we notate the difference? Should we require closed polylines to start and end with the same point? Note that if so we need two different identical points to be part of each point-set, so point-set ordering becomes consequential. Alternatively, we could model polylines and polyarcs as two different shape *types*, where the former (but not the latter) have an implicit edge between their last and first points (again this requires that ordering be fixed). Another possibility is to introduce a "pseudo-coordinate" such as **TikZ**'s "**cycle**", which yields a point duplicating the first in a sequence (how to accommodate those special coordinates, needing contextual interpretation, in a point data-type then becomes a concern for implementations). Finally, a fourth option is to use some sort of flag to denote the intention to treat a given point-set as spanning a closed path rather than open arc (this is potentially more flexible because it does not foreclose options in how to deal with point ordering). For sake of argument, this last approach is taken in the book's demo code.

Another consideration for representing polylines and polyarcs is how to deal with (three or more) colinear points. Technically, a point midway between two other points in the same point-set, where all three lie on the same line, is extraneous. Should points in such circumstances simply be dropped from the annotation, or should constructions involving colinear points be rejected as malformed? This question depends on whether we consider intermediate colinear points to have any semantic value or meaning (despite their apparent mathematical superfluousness).

Consider the following scenario: a user creates and then edits an annotation by dragging handles representing vertices in the annotation-shape. The user might try to visually form to annotation so as to encircle a Region of Interest, let's say. This leaves open the possibility that they may drag a vertex to a point colinear with two adjacent vertices. However, it would not make sense to simply eliminate that vertex, because the user may potentially drag it once again, which could result in the colinearity being broken. Since the annotation might be stored and edited again at some future time (potentially by a different person), one can argue that it is premature to simply drop intermediate colinear points from the point-set. On the other hand, it may be appropriate to *flag* them as superfluous from a mathematical point of view with respect to the annotation's current state (again, see our demo code for a concrete illustration of a point-set data type adopting this approach).

This modeling decision translates to procedural functionality: if extraneous colinear points are permitted to be part of an annotation's point-set but excluded from geometric algorithms, a natural operation to "restrict" the annotation to its geometrically valid point-set would be to offer a procedure which maps the point-set onto a filtered version with no superfluous points. One could similarly implement "normalizing" functions which address concerns such as point-order. In other words, even if the annotation's data model does not explicitly stipulate restrictions or policies for (say) self-crossing point orders, code libraries instantiating the data model could provide procedures to convert more free-form annotations to ones which obey stricter geometric guarantees.

Note in particular that many issues we have described here as *data modeling* questions similarly have procedural ramifications. The issue of colinearity organically gives rise to the notion of normalization procedures which filter out intermediate colinear points. Issues of chirality and self-intersection suggest procedures to order point-sets into clockwise or counter-clockwise sequences (and to invert them so as to switch orientations). Both colinearity and chirality/convexivity come into play with respect to validating point-set representations and/or building point-sets incrementally: what procedures are implemented to add a new point to an existing set, or to initialize a point-set from multiple points *ab initio*? What are those procedures' pre- and post-conditions? For example, should a precondition for initializing a polyline be that the supplied points span a convex region? These are examples of how data-modeling decisions tend to translate to coding requirements, the pattern of intersection between data and code modeling which we discussed

in Chapter 6.

2.2 Annotations with Curved Geometries or Cross-References

The last few paragraphs have focused on annotation-shapes based on straight line segments. Of course, many annotation are better expressed via curved paths, whether open or closed. Some curves can be specified via small point-sets; e.g. ellipses may be defined by asserting their focal points and one point on the curve itself. In general, though, curve-definitions require combining points with scalar magnitudes representing distances, rather than locations. Circles have centers and radii; ellipses have focal points and eccentricities. Circular or elliptical *arcs* can be defined by constructing the full circle/ellipse and then providing start/end angles.

The point here is that annotation data structure may incorporate sets of magnitudes (call them "length-sets") as well as point-sets. Sometimes length-sets can have additional structure; for example, one common strategy for defining ellipsoids (generalizing from 2D ellipses to higher dimensions) is via "covariance matrices," paired with designation of a single (center) point. In that case the length are associated with matrix row/column positions. Length sets might potentially be used for polylines as well as curves: for a regular polygon with n sides and d diagonal, it is probably more convenient to assert the shape's center as a point and the remaining data as lengths (calling n a "length" just in the sense that it is a scalar quantity rather than the location of a point).

Consider also *arrows*, which are often employed as image overlays. Arrows can be a visual device to call attention to a specific image location or region. If arrows are recognized as a distinct annotation-type, then at a minimum one should identify a point which the arrow "targets." Alternatively, we could allow arrows to be paired with other annotations, so that the arrow "points at" their shape. Arrows may also connect two different prior annotations. Factors such as arrows' length and width, or the styling of the arrow "head," may be either visual/presentation details or intrinsic to the arrow-annotation. Assuming at least some data pertaining to the arrows' shape is deemed intrinsic, then this would be another case where length-sets (e.g., arrow length and width) would be needed as part of the shape data. Arrows also illustrate the first point that some annotations may reference *other* annotations, representing additional data fields which are not subsumed under point-sets or length-sets.

Curves which are more complex than circles or ellipses may also be constructed via techniques such as **b-splines** (which use point-sets as "control points" deforming the shape into the desired curve form) or by stipulating a particular genre of curve (e.g. parabolic or sinusoidal). Conceivably, one could introduce a generic "open-curve" shape-type which includes a data field labeling the more specific kind of curve intended. These labels would then signal to the rendering engine that a particular algorithm should be used to generate the curve given the point and length sets included in the annotation data. In this context, geometric data such as bounding boxes, convex hull, deformation measures (how much a shape's area deviates from the smallest circle containing it), calculations as to whether a line segment between two points would intersect the curve, and so forth, would need to provided via algorithms specific to the curve's genre. In this situation the annotation framework might specify the *kinds* of algorithms which need to be available for arbitrary curve-types but leave their implementation open-ended.

To summarize our discussion so far, then, describing annotation-shapes in general requires a number of different data structures which would be intrinsic to annotation data: point-sets; length-sets, or in general collections of magnitudes which might supplement point-sets to uniquely fix a shape; in some cases, designation of *other* annotations referenced by a given annotation (such as an arrow, or, potentially, say, a circle grouping two or more other annotations); and, in some cases, designation of a special-curve genre with support for adding special-purpose rendering algorithms in a modular fashion. As this overview suggests, there are multiple data-structures which need to be representable for general-purpose annotations even when defining the underlying annotation shape, setting aside issues such as text description and image references.

3 Annotations and Image Features

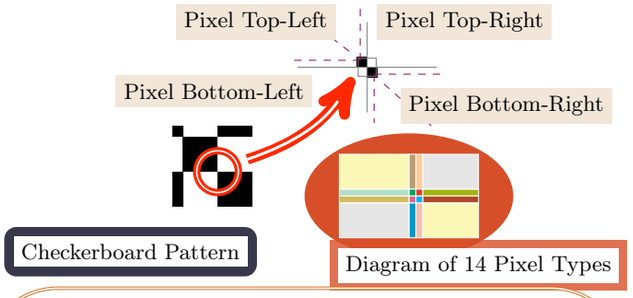
The previous discussion focused on annotations which have a fixed geometric outline, but what about descriptions or markup which extracts information from the image in other ways? For example, a point-set could be employed as a summarial overview of an image without the points being intended as vertices of a convex polygon. Consider an image-processing pipeline to count the number of cells in a Whole Slide Imaging (WSI) view: one could match each cell with a distinct identifying point, then count the number of those points as proxies for the cell themselves. The point-set thereby summarizes the image without describing a particular connected shape. Point-sets could also potentially be used to model textures, diffusion processes, or other visual effects apparent in an image.

As image-processing has become more sophisticated, biomarkers extracted from an image have taken on a wider variety of forms, not only effects such as tumors or legions which can be circumscribed via a closed annotation-shape. For example, image analysis of cancerous tissues or nodules can reveal optical patterns which are indicative of tumor features such as heterogeneity, hypoxia (lack of oxygen), and angiogenesis (proliferation of blood vessels supporting the tumor). Such characteristics affect tumors' malignancy and aggressiveness: in general, hypoxic and heterogeneous tumors are more dangerous and resistant to clinical therapies.

Identifying these signals is a different process than, for in-

Figure 1: Checkerboard Pattern as a (Very Simple) Texture

Color-Change (Gradient) Matrix (for the top-right pixel): $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$



All pixels in this image can be sorted into 14 groups: white pixels surrounded entirely by white; black pixels surrounded by black; white (respectively black) pixels on horizontal or vertical edges between black and white edges; and pixels at black/white corners. Each of these groups can be associated with a distinct gradient matrix showing jumps in color values along eight directions relative to each individual pixel (the diagram above shows some of these groups by assigning them distinct colors, though the actual pixels are only black/white).

For real-life images, checkerboard patterns would presumably only be exact, so the gradient matrices would have entries other than perfect 1s and 0s. However, each matrix could still be clustered into one of the 14 canonical matrices for the pixel groups, allowing the pixels to be classified, and same-group pixels should be arranged according to a pattern matching that diagrammed above. Patterns of groups in the neighborhood of individual points can similarly be gathered into matrices, e.g.:

$$\begin{bmatrix} Group1 & Group2 & Group3 \\ Group1 & (center) & Group3 \\ Group1 & Group2 & Group3 \end{bmatrix}$$

The group-number of the center for each such matrix would predict the surrounding groups, and a secondary matrix could check whether these predictions are accurate (using "1" to mean a correct group, and "0" an unexpected one):

$$\begin{Bmatrix} 1 & 1 & 1 \\ 1 & (center) & 1 \\ 1 & 1 & 1 \end{Bmatrix}$$

The closer these matrices are to having all 1s, adding over all points in the image, measures how closely the image approximates a perfect checkerboard.

This may not be the most efficient algorithm to perform such an analysis; we describe it here simply as an illustration of techniques used for detecting more complex textures.

stance, segmenting a tumor from a background of ordinary tissue; instead, suggestive indications for conditions such as heterogeneity, hypoxia, or angiogenesis involve patterns that can be mathematically extracted from extended regions in an image. For example, [4] document techniques for estimating heterogeneity by partitioning a nodule (as recorded via 2D graphics) into textural segments, where different sectors are isolated by considering the local resemblance of neighborhoods around individual image-points to canonical patterns, such as "harmonic wavelets" (page 4). Similarly, angiogenesis (correlated with tumors' proclivity to expand and co-opt surrounding tissues) can be measured by detecting patterns in nascent blood vessels, which in turn are extracted (in mathematical image processing) by looking for point-neighborhoods which reflect the specific qualities of vascular nodes where a tree-like pattern splits from larger branches to smaller ones (see e.g. [8] and [10], or [20], although the latter documents methods oriented more towards morphology and edge-detection than in the above overview).

In general, these kinds of textural analyses are similar to image segmentation or edge/contour-detection based on colors — where segments are assumed in general to be regions of similar color that are bounded by adjacent (background) regions with different colors; the boundary between a segments and its background is therefore defined by a noticeable displacement in color space — except that the basic analytic units are "textures" rather than colors. Regions of similar colors are quantified by measuring color-distances between different points, which is straightforward because color space is easily metrized into HSV (hue, saturation, value) or RGB (red, green, blue) coordinates. Quantifying textures, however, is more difficult, because texture-data is not manifest directly within individual pixels the way colors are. However, it is possible to mathematically calculate the degree to which the neighborhood of a given pixel approximates what would be expected if the image perfectly matched a particular kind texture around that point (Figure 1 is a very simple depiction of this sort of analysis). Texture can therefore give rise to vectors of "signals" mapping points to one or more magnitudes characterizing the textural pattern around each point, to the degree that this pattern is approximated by the image data. These texture-vectors can then take the place of color vectors (i.e., coordinate triples such as HSV values) for segmentation, edge-detection, and similar image-analysis operations.

Marking the presence of textures (or the fact of an algorithm having identified them), however, makes annotations more complex than regions or patches identified in terms of color alone, because to be thorough the annotation would not only have to identify the relevant region (e.g. a region where a certain texture is evident) but to describe the texture itself. Unlike colors, which can be specified with only three quantities, textures may involve angles, displacements, gradients, and other mathematical parameters. Such parameters can be associated with individual pixels to create image-like data structures within an image-processing pipeline, where pixel-data associating pixels with *colors* is replaced by data involving texture-definitions.

Image-analysis workflows may therefore have several intermediate processing steps defined by midstage images or image-like resources derived from earlier images in the workflow via feature analysis or morphological operators. As such, image processing operations do not always act directly on images themselves; sometimes algorithms are based instead on mathematical complexes derived from the image,

but with their own quantitative properties. For instance, color-valued pixels may be replaced by matrices measuring the gradient of some image-feature field in eight directions around each point (an example would be "Sobel kernels" applied to the image intensity function [9, for instance]). Data structures for describing textures are therefore more complex than for colors, and (more to the point) it is harder to stipulate *a priori* specific forms which these structures would take on. While we can fully capture all digitally reproduceable colors within spaces like HSV, one cannot develop an exhaustive list of all sorts of textures that might be analytically relevant for image-processing; thus the details of texture-data would need to be open-ended for a general-purpose annotation model.

3.1 Specifying Annotations' Roles and Origins

A further detail that should be clarified is that of how image-annotations originate. Sometimes, of course, annotations are manually introduced on images by human users of image-viewing software. On the other hand, automated image segmentation — or similar algorithmic or AI-driven image processing without human intervention — yields partitions of images into regions, or identification of semantically important locations in an image, therefore generating annotations computationally. In short, annotation descriptions should support both human-generated and computer-generated annotations. This becomes complicated, however, when we consider the full range of computerized image-analysis capabilities, such as texture-analysis and similar feature-extraction techniques, which can potentially yield more complex data structures that represent statistical patterns evident in the image (often after mathematical transformations of the underlying image data). Image-processing may yield analyses which overlap with annotation objectives but may not intrinsically produce annotations in the conventional sense. For example, an algorithm to infer the number of nuclei in a cell-scale picture — or (not a biomedical example but useful) assess traffic patterns by counting cars — may rely on statistical analysis of some quantitative image feature (such as "zero-crossings") without in fact producing determinate image segments. As mentioned above, notating textural patterns and feature-vectors along these lines is more complex than simple polygonal or elliptical annotation-shapes.

For a given "semantic" task — that is, an image-processing objective whose end-result is not just image-related data but some empirical observation — image segmentation, or other analyses yielding annotations, are a means to an end: one *way* to count nuclei is to delineate the edges of distinct nuclei in distinct segments, and then count the number of segments which result. However, statistical image-analysis may produce largely accurate results for such semantic tasks, given large image corpora (e.g., estimating traffic flows from highway cameras), without yielding artifacts such as human-visible segment representations. Or, in a different domain, AI-powered analysis of FCS (Flow Cytometry Standard) data could establish a largely accurate count of "events" (i.e., discrete FCS measurements of light-scattering and/or fluorescent properties of cellular-scale entities) without manual "gating" (referring to the conventional practice of scientists using geometric annotations of FCS data-plots to isolate and thereby count different event-types). An AI-powered analysis of image features, or likewise of Flow Cytometry (FCM) data, may yield calculations similar to those which for *human* users are achieved via image segmentation, manual gating, and similar operations which clearly yield annotation data.

The complication arises when AI mechanisms along these lines do not themselves yield results that would normally be considered annotations, but rather yield the desired empirical results for which the annotations would be a preliminary step — e.g., an approximate count of the number of nuclei in a slide-image or cars in a highway photo, without a precise segmentation of the image marking their respective borders.

An image annotation is, among other things, a visual (or viewable) record of some image-processing activity. If a radiologist manually clarifies a report to the effect that a given CT scan shows a tumor by circling the area where the tumor is visible, he or she is using the image annotation to communicate to others the thought-process which motivated the diagnostic conclusion. This is different than an AI-driven processor which would automatically demarcate an image segment outlining the tumor and use geometric properties of that segment to derive a pathological finding. In short, the data conveyed in an annotation — an image segment, rendered precisely, or rendered indirectly via a circle or polygon around the segment — may be *intrinsic* to an image-processing operation: it may be data acquired *at one stage* in an analytic workflow. However, annotations may also be *retroactive*; if a radiologist circles a tumor, he or she has completed (at least mentally) the image analysis, and is using the analysis to summarize what occurred in the course of the analysis. Therefore, any image-processing task can be associated with *ex post facto* annotations which summarize the process even if they are not intrinsic to it.

To continue the example of counting nuclei from a microscopy image (or cars from a traffic camera), an AI-powered observation might be retroactively *justified* by providing a segmentation where the number of cell-segments (likewise car-segments) matches the AI count. In lieu of precise segments, however, it may be simpler to provide location-points for the "geometric center" of each cell (respectively, car), or the points furthest apart in the direction of each car's front-to-back — these may be the statistical signals used for the car-counting process (such orientation-based enumeration makes more sense in the traffic example than the microscopy). Analogously, facial recognition does not need to rely on segmenting out regions (eyes, nose, lips), but rather can be based on distances between individ-

ual points (such as the inner corners of each eye).[6]

But in any case, depending on the analytic algorithm used, it is often possible to identify some spatial/geometric feature or object that can be visualized in the image context, and which summarizes or legitimizes the analytic operation. This summarial data, then, can provide *retrospective* annotations which allow human viewers to understand and review the algorithmic process. In short, simply because image-processing tasks may not generate annotation data as part of their internal activity, it is still possible (and may be desirable) for the software operationalizing these tasks to implement annotation generators, where the resulting annotations document the operations for the scientific record and/or summarize them in GUI objects for the benefit of human viewers.

In general, then, we can distinguish human-generated from computer-generated annotations, and moreover leave open the possibility that some computer-generated annotations are *retrospective*: that instead of being internal to an imaging computation they are indirectly produced subsequent to such a computation, for purposes of documentation and validation. Annotations which are not *retrospective* could be called *internal*, as in, internal to a given image-processing workflow.

Related to the distinction between *internal* and *retrospective* annotations, we can also recognizes a contrast between "immersed" and "descriptive" annotation (these are idiosyncratic terms but they seem appropriate for the contexts involves). An example of an *immersed* annotation might be an image segment, where calculating the boundary of the segment is intrinsic to a specific image-processing objective, whereas an *descriptive* annotation might be an arrow *pointing toward* that segment. Here the descriptive annotation is introduced primarily for the benefit of human viewers.

The immersive/descriptive distinction is not always clear-cut. Consider the following two cases: in one scenario, an image-segmentation routine precisely delineates a region of interest (e.g., an outline of a red bird against blue sky), notating the segmentation result via a two-color-depth transform of the original image. Image-viewing software then shows the segment indirectly by encircling it, producing, in effect, a secondary annotation intended to call attention to the primary (segment) annotation. Here, clearly, the segment itself (encoding by a separate two-toned image) is intrinsic to the original analysis, whereas the secondary annotation has a purely descriptive purpose.

However, consider an alternate scenario where the original segmentation is done with less precision (this may be the case where there is a more muted color differential between foreground and background). Imperfect segmentation may still be adequate for some semantic task (e.g., identifying a bird's species). An analysis could obtain a rough segmentation by marking certain points highlighted by an edge-detector, then protruding the convex hull of the region outward so as to be sure of encompassing the whole bird-segment within a polygon, albeit allowing some background pixels into the polygon as well.

This approximate segment is only an indirect representation of the region of interest (which would be the avian subimage clearly outlined with no background included), but for analytic purposes the rough polygon might be a reasonable substitute for the finer-grained segment, analogous to how a cubic or quartic polynomial may be an adequate approximation to a more complex curve. Depending on how it is used, the approximate segment may therefore be considered merely a visual cue connoting the region of interest, or a significant region in its own right. Such a distinction would, most likely, depend on whether the approximation is used itself as a basis for further analysis, or instead is mostly a presentation device. This usage-context would therefore indicate whether an "imprecise" annotation should be classified as *immersed* or *descriptive*.

All told, then, annotations may be *internal* or *retrospective*, and *immersed* or *descriptive*; *computer-generated* or *human-generated*; and *manual* or *automated*. These distinctions are independent of one another; retrospective annotations for instance could be either immersed or descriptive, and either computer-generated or human-generated. These aspects of annotations and/or groups of annotations can be notated in various ways. One option (adopted by the book's demo code) introduces an *originator* object which provides information about how annotations and/or annotation-groups were created. These objects' data fields can then clarify the roles and mechanisms driving their correlated annotations.

When applicable, annotations' roles are also, for obvious reasons, closely implicated with calculations *about* an image performed with the aid of annotations. As is concretely demonstrated in the context of AIM, annotation data may include quantities such as the length of a line segment or the area of a region. More complex forms of image-features engender more complex calculations; consider fractal dimension estimation in the context of angiogenesis [3, page 4], [19, page 6], or Gradient Vector Flow (GVF) (with applications to image-segmentation in contexts such as diagnostic pathology/oncology) [22, page 85], [21]. In AIM, calculations can be presented via both text descriptions and formulae composed in the Mathematical Markup Language (MathML), but these cannot provide a detailed and machine-readable representation of data structures that may be used generated by computationally intensive feature-extraction methods. This leaves open the question of how to properly encode the broad range of calculations that could potentially be applied within the context of an annotation.

In effect, a general-purpose annotation framework needs to consider how much detail should be represented as one shifts focus from annotations themselves to the segmentation methods, textural features, and biological interpreta-

[6] See e.g. [16], [13], [6], [7], or [14],

tions which enter in play to the degree that *image* details are taken as indicators or warrants for image *biomarkers*. We will consider this question in the next chapter.

References

[1] Greg J. Badros, *et al.*, "A Constraint Extension to Scalable Vector Graphics". In *Proceedings of the 2001 International Conference on the World Wide Web*, pages 489-498. https://constraints.cs.washington.edu/web/csvg-www10.pdf

[2] Enrique Martinez Berti, *et al.*, "Dual Quaternions as Constraints in 4D-DPM Models for Pose Estimation". *Sensors*, Volume 17, 2017, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5579524

[3] Marie-Laure Boizeau, *et al.*, "Automated Image Analysis of In Vitro Angiogenesis Assay". *Journal of Laboratory Automation*, Volume 18, pages 411-415. https://journals.sagepub.com/doi/pdf/10.1177/2211068213495204

[4] Dmitry Cherezov, *et al.*, "Revealing Tumor Habitats from Texture Heterogeneity Analysis for Classification of Lung Cancer Malignancy and Aggressiveness". *Scientific Reports*, Volume 9, 2019. https://www.nature.com/articles/s41598-019-38831-0

[5] Alexandre Carlier, *et al.*, "DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation". In *34th Conference on Neural Information Processing Systems*, Proceedings, 2021. https://proceedings.neurips.cc//paper/2020/file/bcf9d6bd14a2095866ce8c950b702341-Paper.pdf

[6] Wei-Lun Chao, "Face Recognition". http://disp.ee.ntu.edu.tw/~pujols/Face%20Recognition-survey.pdf

[7] Yueqi Duan, et al., "Topology Preserving Graph Matching for Partial Face Recognition". In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, 2017. http://ivg.au.tsinghua.edu.cn/people/Yueqi_Duan/ICME17_Topology%20Preserving%20Graph%20Matching%20for%20Partial%20Face%20Recognition.pdf

[8] Charalampos N. Doukas, *et al.*, "Automated Angiogenesis Quantification through Advanced Image Processing Techniques". https://pubmed.ncbi.nlm.nih.gov/17946107

[9] Panteha Eftekhar, "Comparative Study of Edge Detection Algorithms". Thesis, California State University Northridge, 2019. https://scholarworks.calstate.edu/downloads/ff365796m

[10] Josef Ehling, *et al.*, "Micro-CT Imaging of Tumor Angiogenesis: Quantitative Measures Describing Micromorphology and Vascularization". *Biophysical Imaging and Computational Biology*, Volume 184, Number 2 (2014), pages 431-441. https://www.sciencedirect.com/science/article/pii/S0002944013007268

[11] Georg Fuchs, *et al.*, "Progressive imagery with scalable vector graphics". In *SPIE 7881, Multimedia on Mobile Devices*, Proceedings, 2011. https://vcg.informatik.uni-rostock.de/~schumann/papers/2010+/Rosenbaum-EI11b.pdf

[12] John Gustafson and Isaac Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic", http://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf

[13] Gary B. Huang, *et al.*, "Towards Unconstrained Face Recognition". In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Proceedings. http://vis-www.cs.umass.edu/papers/unconstrained_face_workshop.pdf

[14] Patchaiah Kalaiselvi and Sivasamy Nithya, "Face Recognition System under Varying Lighting Conditions". In *IOSR Journal of Computer Engineering*, Volume 14, Issue 3 (Sep.-Oct. 2013), pages 79-88. http://www.iosrjournals.org/iosr-jce/papers/Vol14-issue3/M01437988.pdf

[15] Yang Li, *et al.*, "4DComplete: Non-Rigid Motion Estimation Beyond the Observable Surface". https://arxiv.org/abs/2105.01905

[16] Feng Lu, *et al.*, "Adaptive Linear Regression for Appearance-Based Gaze Estimation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 13, Number 10 (2004), pages 2033-2046. https://www.ncbi.nlm.nih.gov/pubmed/26352633

[17] Daniel L. Rubin, *et al.*, "Medical Imaging on the Semantic Web: Annotation and Image Markup". http://cedarweb.vsp.ucar.edu/wiki/images/d/d9/R_19.pdf

[18] Michail Schwab, *et al.*, "Scalable Scalable Vector Graphics: Automatic Translation of Interactive SVGs to a Multithread VDOM for Fast Rendering". *Micromachines*, Volume 11, Issue 7 (2020). https://ieeexplore.ieee.org/document/9354592

[19] Matvey Sprindzuk, *et al.*, "Computer-aided Image Processing of Angiogenic Histological Samples in Ovarian Cancer" https://core.ac.uk/download/pdf/8701857.pdf

[20] Ioannis Valavanis, *et al.*, "A Novel Image Analysis Methodology for the Evaluation of Angiogenesis in Matrigel Assays and Screening of Angiogenesis-Modulating Compounds". https://link.springer.com/content/pdf/10.1007%2F978-3-319-23868-5_5.pdf

[21] Chenyang Xu and Jerry L. Prince, "Snakes, Shapes, and Gradient Vector Flow". *Transactions on Image Processing*, Volume 7, Number 3 (1998), pages 359-369. https://pubmed.ncbi.nlm.nih.gov/18276256/

[22] Lin Yang and David J. Foran, "Deformable Models and their Application in Segmentation of Imaged Pathology Specimens". In Jasjit S. Suri and Aly A. Farag, *Deformable Models* (Springer 2007), pages 75-94. https://link.springer.com/chapter/10.1007/978-0-387-68343-0_3