ConceptsDB Image and Data Formats

New Representations for Storing and Accessing Image Data, Metadata, and Multi-Value Database Assets

Description

ConceptsDB is a multimedia database engine optimized for storing and analyzing large and/or multidimensional resources and data structures. **ConceptsDB** is built to offer programmers a smoother development process for working with data structures and resources which, by virtue of size or internal complexity, would demand multiple transformation stages when being persisted in traditional database systems.

One priority when engineering **ConceptsDB** was building a framework for performing image-analysis over (potentially large) collections of related images, i.e., image series. Analyzing image series introduces logistical complexities above and beyond the steps required to analyze single images themselves. A second priority has been supporting interactive and interlinked images and image series. This includes facilitating the implementation of **GUI** components which render images in an interactive manner, ensuring that annotations, color/geometric data, image-to-series relations, and other image details are communicated to users via intuitive presentations which comport with interactive software expectations (mouseover effects, secondary dialog windows for temporary information displays, and so forth). In addition, images may be interactive in conjunction with image series insofar as links may exist between disparate images or segments/regions thereof (see page three for a further discussion on cross-image linking).

With respect to image analysis, separate and apart from image-processing pipelines — which may subject a single image to a series of transformation steps, followed by statistical analysis of the image as it appears at one or more of these pipeline stages — image analysis calls for workflows that prepare the image for processing (acquiring it from some source and translating image-data to a format optimized for processing) and then, subsequent to processing, aggregate statistical data into feature-vectors and observations. In the context of image series, these workflow steps (the preparatory and follow-up logistics as well as image-processing pipelines themselves) need to be repeated for multiple images all together, and post-processing results from single images typically must be merged into a series-level feature-space. Often a small set of prototype images or "training data" are selected as reference-points for establishing workflows, with parameters defined relative to the prototype images then being applied to all other images in a series. Analyses may be run multiple times with different choices of initial parameters.

Although the stages required to marshal image data into workflows — and handle feature vectors coming out of workflows — can occupy a significant portion of image-processing code, patterns for pre- and post-processing data have been less rigorously studied than image-analysis/Computer Vision themselves. As a consequence, one can observe a tendency toward computational bottlenecks, with projects comment implementing pre- and post-processing code in an ad-hoc fashion. **ConceptsDB** tries to make this process more rigorous by establishing systematic protocols for defining workflows and then applying them to image-series. As such, **ConceptsDB** can shorten the development time for software components that integrate image-analysis code with database/application environments where images-series are acquired/maintained, and analytic results are mined for actionable data.

To promote more efficient application/database development as outlined above, **ConceptsDB** introduces new formats for representing images and image-annotations, as well as general-purpose data structures needed for reporting feature-vectors and observations, as well as logical interrelationships among images which are intrinsic to different kinds of image-series (temporal relations within time-series, geographic and/or spatial relations among images sampled from satellite or Panoramic image acquisitions, and so forth).

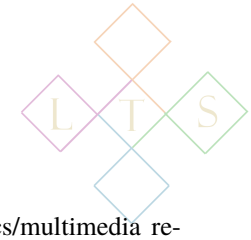
As presented in the accompanying White Paper and slide deck (both of which highlight the important **ConceptsDB** components in greater detail), the new formats introduced in the **ConceptsDB** environment include the following:

- The "**XCSD**" Image format: **XCSD** is a **2D** representation optimized for image-processing, analogous to **CvMat** (from **OpenCV**) and **ITK**-Image (from **ITK**, the Insight Toolkit). In contrast to existing formats, **XCSD** is engineered with greater emphasis on multi-scale analysis and on simplifying transformations between different color models. Many algorithms are provided as intrinsic features of **XCSD** images that in other contexts would need to be implemented separately (external to the image data proper, which complicates image-analysis code). In this manner **XCSD** can simplify the specification and implementation of image-processing pipelines. Specific features of **XCSD** conducive to pipeline engineering are discussed in the White Paper.

- **ConceptsDB** annotations and image-tagging: **ConceptsDB** introduces new annotation formats and **GUI** components for creating annotations via interactive image-view portals. In general, **GUI** classes provided as part of the **ConceptsDB** code base recognize multi-stage annotation definitions where different sequences of user actions and mouse gestures create annotations of varying shapes and structures. **ConceptsDB** recognizes a wider range of annotation styles than traditional image-tagging environments.

- **ConceptsDB** Collections: **ConceptsDB** implements a "multi-value" database model, where data collections (typically defined as dynamically resizable lists or sets of similarly-typed values) are stored and accessed/queried as first-class object in the database (in contrast to relational/**SQL** models — and also many **NoSQL** database architectures — which only support multi-value collections indirectly, complicating the steps which must be coded to manipulate data collections). Streamlining the persistence and querying of multi-valued data structures makes it easier to implement code where image feature-vectors and other image-processing details are integrated with the database instances where images themselves are stored.

In contrast to most image formats (linked to database engines only as external files or as opaque binary data that cannot be examined through database queries), **XCSD** images may be stored and analyzed directly inside **ConceptsDB** database instances, if desired. In some contexts this approach is more efficient than pulling raw image data from a database, executing workflows against those images, and then exporting the results back to the database. In short, **ConceptsDB** and **XCSD** have been designed together with the goal of systematically modeling image semantics, workflows, and color/segmentation data (geometric, textural, diffusional/directional, and color-statistics feature spaces, applicable on multiple scales from image-series to single images, segments, and superpixels) via patterns amenable to database persistence and query/retrieval. (More information about the **XCSD** format and database architecture that supports in-database processing is available in the White Paper.)



Interactive/Interconnected Image Series

Given that a multi-media database will intrinsically store collections of images (or other graphics/multimedia resources), it is important for users to be able to model and access any interconnections between images/assets which are relevant to their project. Database queries and image displays based on **XCSD** identify cross-resource links and allow users to browse through image series based on logical interconnections. For example, one portion (area/region) of an image may represent or be associated with other images — e.g., **PDF** files representing architectural floor plans may link to photographs documenting the installation of building fixtures, or to dashboards for Cyber-Physical actuators. Likewise, the photograph of one room in a building can link to more focused pictures of individual features inside the room; similarly, single-room panoramic photos (seen through a **3D** tour) can embed links to **2D** photographs.

The **XCSD** format has several novel or nonstandard features which are rooted in software engineering principles for interactive images. In particular, **XCSD** represents multiple "tiers" of image detail, and typically encodes images through medium-scale boxes of memory-contiguous pixels, called "tierboxes" (in **XCSD** code). Tierboxes form the basic scaffolding through which memory layout, color data, interactive features, and user-event signals are organized. The tierbox system provides several benefits, including (1) pre-encoding thumbnail images for database queries and multi-image displays; (2) precalculating pixel groups for algorithms (such as local color histograms) that operate on "tiles" spanning an image rather than its global dimensions; (3) organizing pixel data such that boxes at different tiers (canonically 3×3, 9×9, and 27×27) are stored via contiguous pixel arrays and that pixel data in-memory begins at the center of an image and expands outward; (4) image-processing algorithms can be engineered, tested, and examined on a small scale (e.g. 27×27) even if in deployment they are not tile-based; (5) each image encodes precalculated data about the distance and orientation of tierboxes vis-à-vis the image center, which can be important for some algorithms; (6) tierbox centers (or box centers at finer tiers) provide a built-in sampling option for algorithms seeded by a regular lattice subset of pixel data; and (7) each tierbox provides a locus for event/signal processing and interactive segmentation.

In an interactive **GUI**, tierboxes can be encoded with limited memory capacity, allowing for responsive user controls. For example, mouseovers can be highlighted by manipulating the appearance of pixels in a single tierbox (whereas it would be infeasible to implement mouseovers at a finer-grained scale, e.g., individual pixels themselves). For tierboxes which lie within a region segmented via image analysis, or as part of cross-image navigation, mouseovers (and related click/hold styling) can be given a distinct visual appearance, so that tierbox-level styling can provide the primary visual cues conveying information about image layout, such as the boundaries of regions which link to other images.

XCSD introduces a nonstandard Manhattan/Chebychev distance representation (combining the familiar Manhattan and Chebychev metrics) as well as a diverse range of coordinate systems and axis labels to support computations related to pixel-level as well as tierbox-level (and intermediate) distance, color means (via different averaging equations), and geometry/isotropy computations. Instead of raw numbers, **XCSD** code uses (and exposes data through) labeled coordinates with restricted numeric ranges (and no automated conversions), which helps to clarify the geometric meaning of coordinate values. The **XCSD** **C++** classes also provide numerous built-in functions for initializing coordinates appropriate to each image, accommodating details such as landscape versus portrait orientation and odd/even combinations in horizontal/vertical tierbox counts, as well as center-to-periphery subdivision indexing (the complexity of these computations is encapsulated by a suite of convenient utility functions). In general, **XCSD** provides a distinctive mathematical environment in which to perform image-analysis calculations, with an emphasis on color-examination and multi-scale computations that are not natively supported by other image formats.