Project acronym: LADIO Project number: 731970 Work package: Deliverable number and name: D2.2: List of file formats to use unchanged and recommendation for filling gaps	Title: Data Model and API Work Package: WP2 Version: 1 Date: May 31, 2017 Author: Carsten Griwodz
Type: [X] Report [ ] Demonstrator, pilot, prototype [ ] Website, patent filings, videos, etc. [ ] Other	Co-Author(s): Benoit Maujean, Pierre Gurdjos, Jonas Markussen, Tomas Pajdla, Thomas Eskénazi To: Albert Gauthier, Project Officer
Status: [ ] Draft [ ] To be reviewed [ ] Proposal [X] Final / Released to EC	Confidentiality: [X] PU – Public [ ] CO – Confidential [ ] CL - Classified
Revision: Final	
Contents: Deliverable 2.2. List of file formats to use unchanged and recommendation for filling gaps.	

Table of contents

Introduction

Objectives

The objective of WP2 is to ensure that the structure and relations of the data collected during the work on set is maintained, along with information about position on a common timeline, spatial position and orientation of recording devices. Task 2.2 is concerned with the file formats and database formats used by LADIO for distributed storage of data (such as audio and video, LiDAR, 360 images, meshes, textures, models) as well as in-memory data structures.

This starts with a review of existing file formats that are relevant for LADIO partners and identify formats that can be adopted unchanged. Particular attention should be paid to the storage of LiDAR data as well as the data recorded by 360 cameras.

Initial reflections

The classical approach to data management distinguishes between so-called flat files, structured in directories, and database, which was arranged as a collection of tables containing relational information. Data management for these two kinds of store has always been quite different, with repeated in-roads taken by specialists in the one field to occupy also the role of the other. Success in either direction of integration has been claimed, but it has not materialized in any concrete systems. Search engines operating on large data collections have entered this competition rather late, yielding performance increases in totally or partially unstructured data.

In the context of the LADIO Deliverable 2.2, this discussion is interesting because LADIO require data management of large amounts of bulk data whose data is partially time-dependent content, discrete content, as well as metadata providing additional information about both kinds of content. Moreover, it is an essential goal for LADIO to maintain various dimensions of relation between data and metadata that is collected and managed by LADIO components.

Although the relations between layers of content that have been described in Deliverable 2.1 follow a relatively small number of concepts, these have to be mapped into storage. Obviously, we could choose to store only raw media data using an appropriate compression method, and store all relational information in flat metadata files in XML that are derived directly from the structure proposed by D2.1.

While we do actually propose this storage format for the interchange of data between different commercial actors, we do not consider it appropriate for all data that is generated. In particular, LADIO WPs 3 and 4 are concerned with 3D reconstruction, the accuracy of 3D reconstruction, and the experimental addition of multibody structures. Algorithms that LADIO uses for these steps have multiple stages, and efficient operation requires a distribution across several computers, as well as between CPUs and GPUs. This implies that LADIO requires efficient intermediate storage, which is not constrained by standards compliance since it is only meant for efficient interchange between the internal stages of LADIO software.

Compared to the goals of the task, the progress of LADIO has led to priorities for Task 2.2 that are somewhat different from the original plans.

1. LADIO has not chosen a specific LiDAR for the project. While we are still waiting for the availability of an affordable LiDAR that can fulfill the needs of the project, we are working with

rented hardware. This means that we are not able to determine a specific recording format for LiDAR data, because this differs between providers.

2. The experience with 360 cameras has led to the understanding that the best way of using 360 cameras in LADIO would actually consist of processing the panorama streams that are typically delivered by today's 360 cameras. Since all 360 cameras on the market consist of groups of separate cameras that are hardware-synchronized, it is actually most desirable for LADIO to retrieve the individual, synchronized video streams for each camera. These cameras can then be processed as a permanently fixed rig of synchronized cameras, and raw data can be processed, yielding higher accuracy. The current expectation is that the 360 cameras used like this can provide information beyond the tracking of lighting conditions on set, which was initially expected as the only role for these cameras.
3. The accuracy of reconstructed 3D points, lines and surfaces has received much more attention that expect, especially due to the ongoing work on Task 4.1. The number of interconnected stages in CMPMVS (our multi-view reconstruction software), and the number of choices that can be made in terms of algorithms and scalability features, means that assessing accuracy is more important than expected for initial reconstructions, final reconstructions, as well as the use of such reconstructions for camera localization and tracking. We have therefore put more effort into the exploration of our options for maintain this information and making it accessible.

Video formats

The cameras that are used in everyday shootings in LADIO comprise first, the main cameras, second, the witness cameras that we use for tracking of the main camera, and third, further support cameras that we introduce newly in LADIO.

The first and second cameras are special cases, where we require high-quality information because of their relevance for 3D reconstruction, not to mention the final product of the film. Beyond these two cameras, who special formats we must support, we have to consider the video standards that we can expect from other devices. Today's primary video formats for general use are in the MPEG family of codecs. While H.264 is the consumer format of the day, HEVC has caught up very quickly and dominates the professional world. However, MPEG standards provide a variety of profiles and levels, and the transfer between the film set and post-production stages must be compatible for both main camera recordings and supporting equipment.

In this section, we explore the coding formats including their profiles and levels that we expect to see in the near future.

Main cameras

The main camera brands that we consider in LADIO are Arri, Red and Black Magic. From all of these models, we expect raw frames that are sent over SDI. For Deliverable 1.1, it is important to note that all brands provide meta information in horizontal and vertical blanking intervals in SDI, however, where and how this happens is entirely proprietary.

Witness cameras

The witness cameras that we have used in LADIO so far are

- PointGrey Flea3 USB3 greyscale camera using rolling shutter with global reset, resolution 2080x1552 [Flea3]
- DO3Think M2S120M USB2 greyscale camera with global shutter, resolution 1280x960 [M2S120M]

We have earlier experience with Basler GBEthernet color 1K and 2K cameras with rolling shutter and IDS USB3 color 4K cameras with rolling shutter.

All of these potential witness cameras allow software and hardware shutter synchronization and deliver uncompressed video frames that require a proprietary host-side driver for control and interpretation. The color cameras did furthermore provide alternative representations as Bayer patterns, compressed Bayer patterns, 3-channel YUV representations and grayscale representations, where the conversion was performed on the camera before sending to the controlling host computer. The set of cameras that are currently used for LADIO are grayscale cameras where such considerations are of no relevance.

H.264

H.264 [ISO2005] is a video codec that was standardized by ITU.T and co-standardized as part 10 of ISO/IEC MPEG-4 under the name Advanced Video Codec (AVC), whose original charter was to standardized a new low-resolution codec. H.264 turned out to deliver a better compression ratio than competing codecs in the MPEG philosophy. It was meant for resolutions up to HD, but has frequently been used for resolutions up to 4K. Encoding and decoding it is hardware-supported on a large range of devices, including mobile phones, CPUs and GPUs. It has been adopted by the Digital Video Broadcast (DVB) consortium in their standardization of terrestrial, cable and satellite transmission, replacing MPEG-2 for video encoding. MPEG-4 has also standardized scaling extensions under the label SVC (scalable video codec).

H.264 is not a standard choice for intermediate formats, due to its very lossy compression. However, LADIO expects to use small devices on film set to records additional information. These small devices includes mobile phones and tables whose primary codec choice is H.264. Also the very frequently used GoPro HERO cameras [HERO] and drones including the whole range of DJI products (Phantom 3 4K [Phan3], Phantom 4 [Phan4]) and Parrot Bebop [Parrot] use H.264.

HEVC

High Efficiency Video Coding (HEVC) [ISO2015] is a video codec that was standardized by ITU.T and co-standardized as part 2 of ISO/IEC MPEG-H. There are standardized extensions for multiview coding, high dynamic range, stereoscopy and scalability. HEVC has even higher compression efficiency than H.264, but this comes at the price of higher computing power requirements on the encoding side and larger memory requirements on the decoding side. To account for the fact that higher computing power is no longer achieved by increasing the speed of processors but by increasing the number of processors in computing devices, HEVC revisited numerous assumptions of H.264, and has dropped some impractical dependencies that prevented parallelism.

To account for parallel processing, HEVC extended the notion of slice that was already known in the JPEG image compression format. Since JPEG, frame could be divided into independently encoded horizontal pixels stripes. MPEG adopted this ability with MPEG-1, allowing interframe compression only between identical slices of subsequent frames. HEVC adds vertical slicing, effectively leading to the ability of dividing a video into a checkerboard of independently encoded regions, where each region forms a tile.

Although this was meant to increase the use of parallel processing, it does also provide isolation between tiles, which solves also the HEVC charter's demand for supporting CubeMaps. This deliverable discusses CubeMaps in the context of 360 cameras, where we propose them as the preferred way of storing 360 video in those cases where we cannot extract video streams from the 360 camera's separate sensors in separate streams.

ProRes

Apple ProRes [Apple2017] is a lossy video compression format developed by Apple for use in post-production. It supports up to 8K video. It is not intended as a format for the final production of any film, but for use as an intermediate format between stages of a production. It should be less costly in terms of processing power to compress, and maintain higher quality than the MPEG family of codecs. While not compressing as efficiently, it should still save considerable storage space compared to raw video.

As the name implies, ProRes 422 performs 4:2:2 subsampling, using only intra-compression. To maintain the color depth of many cameras, it uses a 10-bit depth for all layers. ProRes compression uses discrete cosine transformation (DCT) of a YUV color space without fixed length restrictions, and encoding has variable bitrate (VBR) as a result. ProRes 4444 and ProRes 4444 XQ are variants of ProRes without subsampling that also support an alpha channel. The luma and chroma plans can have up to 12 bits depth, the alpha channel up to 16.

ProRes has wide compatibility and has its market in feature film production, with highest quality requirements. Using a ProRes encoder requires a licence, but decoding is free. It can be decoded using a Quicktime plugin, but other compatible decoders exist.

DNxHD

Avid's DNxHD [Avid2012] is a competitor to ProRes that can perform lossy compression of video at 8 or 10-bit depths for editing processes. It has been accepted as a VC-3 standard by SMPTE. It is intended for video resolutions up to 4K and can compress without subsampling at 4:4:4 or with 4:2:2 subsampling. In spite of being standardized, it has so far remained fairly limited to Avid products. It's compression method reminds of JPEG, combining DCT and run-length encoding.

DNxHD is oriented towards the TV market, where it can then be used on shooting set. It is not usually used for feature films. It is frequently used by editors in conjunction with Avid software.

360 cameras

The concept of the 360 camera is not clearly defined, since the name does not imply whether the camera should record 360 degrees in a single plane, whether it should record an entire sphere surrounding the camera, or anything in between these two options. The 360 cameras are unified in their attempt to provide a single 2D video stream that contains the entire 360 view and compresses it using one of the standard 2D video encoding methods. However, the different camera configurations (or rigs) lead to a wide variety of geometric figure that may be mapped into the 2D space of a regular video. Figure 1 shows the main examples of such mappings.

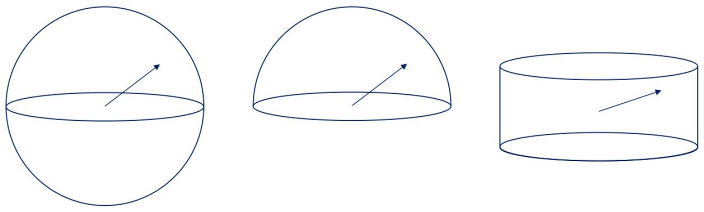
Figure 1: typical pixel mappings of 360 cameras

The majority of 360-cameras deliver a single videos stream of frames that are stitched from several physical cabinets in a single housing. The stitching process is usually straightforward, since camera manufacturers make assumptions about the volumes that are recorded by their 360 cameras and the distances from scenes at which they are used. Consequently, the assumption is made that the camera centers are quasi identical for all cameras of the 360 rig, making stitching trivial.

In LADIO, we prefer for nearly all tasks relevant for post-production to treat the separate cameras of the 360 cam as separate streams with static rig information and perfect sync. The ability to extract streams separately from these rigs means that we do not suffer from the accuracy reduction that is inherent in all space-efficient projections due to their need of interpolating target pixels. The benefit of this solution becomes obvious in the discussion of projections below.

However, there are situations where visual inspection of surroundings of an entire area is desirable. We expect that this does not concerns automatized inspections, but that a person of the team on set wants to verify that a particular part of the set was placed accurately, or search for the reason for a problem. This can be dealt with using today's most popular API, the OSC (Open Spherical Camera) API [OSC2017].

Projections

The big challenge with this approach is the pixel coverage of the mapped image. While there are quite many geographical mappings that maintain pixel coverage in mapping a sphere or dome to a flat map, they are either distorting proportions or they are discontinuous. For example, Apian's projection (Figure 2) can unfold a sphere into an arbitrary number of slices (the reverse is used in stitching globes from fabric), with quite accurate pixel sizes. Storing these pixels into a 2D frame, however, leaves gaps of varying size. It is questionable how typical video compression algorithms handle these patterns.

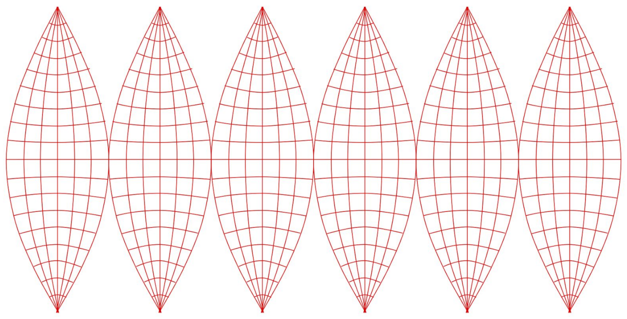
Figure 2: Apian's projection unfolds the globe

Alternatively, projections like the Lambert cylindrical equal-area projection (Figure 3) fill the space of the 2D frames very well, and maintain the same ratio between pixel size and coverage area on a sphere. However, while height and width are symmetrical around the equator, height is traded for width towards the pole, leading to huge distortions.

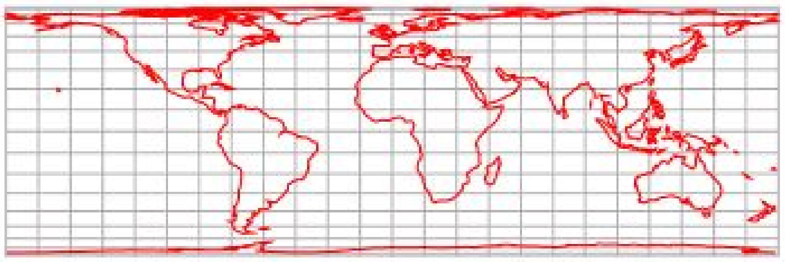

Figure 3: Lambert cylindrical equal-area projection (generated by Wolfram Alpha)

The typical approach that is taken by today’s 360 cams is, however, the plain equirectangular projection (Figure 4). It is simple to implement by rays from a cylinder surrounding a sphere onto the sphere in a plane parallel to the equator plane. This defines the pixels on the cylinder, which is then unrolled onto a 2D frame. The height coverage of all pixels is maintained, however, the width coverage of the projected pixels grows towards the poles, with the final pixel lines representing the pole pixel (if present) covering identical 360 degrees in all pixels of the line.

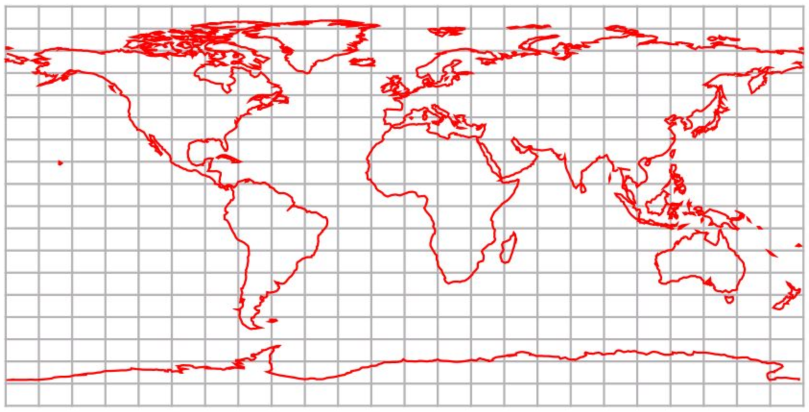

Figure 4: equirectangular projection (generated by Wolfram Alpha)

In computer graphics, a different goal has been pursued, namely the efficient mapping of data into a viewpoint. For this reason, CubeMaps (Figure 5) were developed by Greene [Gre1986]. This became extremely interesting because it was introduced into hardware, as discussed by Voorhies and Foran [VF1994]. The efficient rendering of CubeMaps is not only a feature of high-end graphics card, but it is also part of the specification for low-end hardware, OpenGL ES [Paw2011].

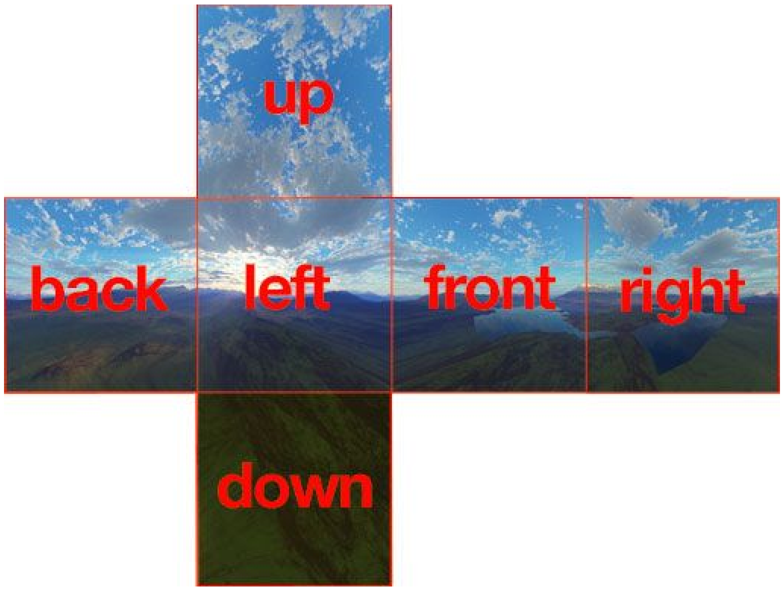

Figure 5: CubeMap of Earth (by Arieee in Wikimedia commons - own work, CC BY-SA 3.0)

Although CubeMaps are increasingly distorted from the center to the edge points of each face, they do not have the singularity at the poles that are exhibited by equirectangular projection. Although the pixel aspect ratio in CubeMaps is distorted in a non-linear manner, this effect is weaker than in the Lambert projection. And finally, CubeMaps that are stored in a reasonable manner, either in 6 separate streams or in an appropriately sliced HEVC stream, allow much stronger data compression than an Apian projection into a rectangular frame. These considerations combined with the efficient hardware-supported accessibility of spherical positions from CubeMaps even on low-end GPUs means that they are the preferred storage format for LADIO in all cases where independent streams from each camera of the 360 camera's rig is not available.

LiDAR

Light Detection and Ranging (LiDAR) is a method of scanning an area that sends laser pulses into a target direction and using a synchronized sensor to measure the time until the reflection of the pulse is seen.

The width of layouts, data formats and scanning patterns differs extremely among the LiDARs that are available on the market, and they range in scope and size from satellite-mounted systems to table scanners for small 3D objects.

It has been used for a while in the film industry to create a highly accurate spatial representation of a film set to make it available for the addition of visual effects. Among the advantages of LiDAR scanning compared to image-based 3D reconstruction is its provision of scale-accurate models, and the resistance to reflective surfaces. Up to now, LiDAR scanning services have mostly been offered by specialized companies that use high-end terrestrial scanners, scan the set in 360 degrees from a small number of fixed positions, and deliver a "cleaned" point cloud and possibly, a mesh as result of their service [Ed2015,Se2012]. Our knowledge at the time of writing the LADIO proposal was aligned with this scenario, although we had the ambition of importing the raw data, point clouds of a static scene that would not include any timing information, into our own processing chains.

It is generally assumed that this scan would be black-and-white, although some LiDARs are capable of measuring a color value. Since the LiDAR itself lights the measured pixel, this is obviously not identical to the color value that a set of cameras would observe from the same point in space, although the difference may only be in luminance.

It is an ambition of LADIO to use the data recorded from several LiDAR scans as verification and refinement for image-based 3D reconstructions. Although the accuracy work in LADIO aims at reducing spurious height errors in the image-based estimation of point positions on flat surfaces, it is obvious that LiDAR data would provide more accurate results. Furthermore, scene reconstructions can be helped immensely by determining the spatial structure of reflective surfaces.

LADIO will use terrestrial LiDaR i.e., systems that typically scan in a vertical direction while rotating about a vertical axis. There are two common situations: (1) the recorder is at a stationary terrestrial location and (2) LiDAR data are continuously acquired from a moving recorder, which can be a typical *modus operandi* in LADIO. LIDAR data collections can be very large and their visualisation can easily become impossible due to memory limitations. The convention is to split LIDAR datasets into subsets of a maximum size and then processing large datasets a subset at a time. This convention becomes mandatory regarding the situation (2).

File format

The exact way in which LiDAR data will be matched with image-based reconstruction is investigated in work package 3, but it is obvious that final information will pass a point cloud stage. For the first recording and storage of LiDAR data, LADIO adopts a format that has first been adopted by the American Society for Photogrammetry and Remote Sensing (ASPRS) in 2003, most recently updated in 2013. This format is called the LASer (LAS) file format [ASPRS2013], and today, the majority of applications processing aerial LIDAR store data in this format.

The intention of the LAS (binary) data format is to provide an open format that allows different LIDAR hardware and software tools to output data in a common format. It sounds appropriate to use the LAS format for terrestrial LiDaR data as it is possible to store much more information than just the point XYZ-position, including the per-point reflectivity value and the per-point accuracy value. The format contains binary data consisting of a public header block, any number of (optional) Variable Length Records (VLRs), the Point (XYZ) Data Records, and any number of (optional) Extended Variable Length Records (EVLRs). All data is stored in little-endian format. The public header block contains generic data such as point numbers and point data bounds.

It is worth noting that while inaccuracy of every single measurement is probably limited due to the small distances expected from most film sets and the high sample rate of the LiDAR, it also becomes extremely important to understand the movement of the recorder as soon as LADIO supports moving recorders, since it is going to be obvious that every sample is recorded from a different camera center position. Accurately recording the movement path of the LiDAR itself becomes a priority for accurately locating the samples in 3D space.

Software availability

There is a LAS library called libLAS that is available under the terms of the BSD License on the website www.liblas.org [Liblas]. However, ASPRS states that this library is near completely superseded by the LASlib, written by Martin Isenburg and published as a package under LGPL license on his web page [LASlib].

It would be preferable for LADIO to adopt and, if necessary, extend the older libLAS, due to our commitment to the MPL.

Products

LADIO has currently gained some experience with Velodyne LiDAR, specifically the PUCK [PUCK]. The PUCK scans 300 000 points per second and has a maximum range of 100 meters. Both its position and internal timing is provided by a GPS receiver (Velodyne's or a 3rd party receiver). The PUCK can yield two outputs per scanned point, providing the distance at the strongest return signal (the distance of the largest area covered by the measurement ray), and the output at the last return signal (ie. the longest distance that could be seen in the measurement ray). Each measurement is associated with a reflectivity value. Vertically, it has a 20 degree field of view, while it can either operate horizontally in 360 degree mode, or an alternative 20 degree mode, which increases the resolution in the target direction considerably. Its accuracy is 0.18 degrees horizontal and 0.07 degrees vertical. It delivers raw samples as UDP packets over Ethernet to a receiver, which in the case of LADIO is a MiniBox.

The GeoSlam ZEB1 [ZEB1] targets measurement in buildings and other indoor spaces. It is a hand-held LiDAR scanner that claims to reach 43 200 points per second. Its range is 15 meters outdoors and 30 meters indoors. The accuracy is high at 0.06 minutes. Data has to be downloaded from a memory card. The interesting feature of the ZEB1 is its ability to work in environments where GPS would not be available of accurate, and uses self-registration instead.

The LeddarTech Vu8 [Vu8] is a LiDAR that is attractive for the LADIO project because the small size and low price of the LiDAR module. It is a member of the new generation LiDARs, whose performance is in the same range of the PUCK's generation, but much easier to acquire and integrate. It has no moving parts and is targeted the self-driving car integration as its primary market. The Vu8 has a range of 215 meters, and a selection of optics that must be installed to choose between 20, 48 or 100 degrees horizontal field of view and 0.3 or 3 degrees vertical field of view. It claims an "up to 100 Hz" refresh rate, reaching 480 000 points per second. Interestingly, it does not scan vertical pixels, but uses a diffuser to sample the entire vertical field of view at once. The accuracy statements about the Vu8 are obscure, claiming 5 cm accuracy, 6 mm distance precision, but only 10 mm distance resolution. The Vu8 delivers data via USB from an SPI or RS-485 connector.

Also Osram has developed a LiDAR package for integrators [Osram]. The Osram 4-channel LiDAR has a range of 200 meters, 120 degrees horizontal and 20 degree vertical field of view, and a resolution of 0.1 degree horizontal and 0.5 degree vertical. It is not supposed to reach the market before 2018.

Point clouds, meshes, textures, models

The integration of real and virtual elements in LADIO requires that the ability to exchange data between systems that generate information from recorded real-world information as well a generated virtual data. In stages of post-production, information that has been recorded from real-world scenes has to be converted into entire scene descriptions in formats that are also useful for a tight integration with virtual elements.

We are also maintaining data that creates virtual representation of real scenes, which are potentially large, and real assets that may be isolated from large scenes and that are potentially small. These virtual representations are not necessarily disconnected from the original real-world recordings, we may need to maintain a stable relation between these two representations to perform essential LADIO functions.

One of the example for the latter is the feature-based tracking of camera poses, which works by identifying features point in a frame of the tracked camera, selecting from a (large) database images that have been used for 3D reconstruction and that have a strong overlap with the newly recorded frame, and compute the camera pose from the location of these images in a reconstructed 3D space.

Furthermore, LADIO expects that an increasing amount of post-production activity will be moved to the film set, initiating the integration of virtual elements either in real-time for immediate review by the photographer and director, or immediately after shooting, allowing for discussions and corrections before the film set has to be abandoned.

Consequently, LADIO assumes that 3D information (including fine-grained 3D reconstructions of an entire scene) must be generated in real-time or near real-time. This does currently still require a trade-off between speed and accuracy, especially since network conditions on film sets do frequently preclude the use of cloud resources as additional computing power. This compels LADIO to maintain information about the accuracy at which 3D information has been extracted from real-world shots, and how this translates into errors when virtual objects are integrated.

To enable this, LADIO is looking for means of storing data that allows to maintain relations between real-world recordings, virtual data and reconstructed data, and that is useful for post-processing in further stages of a production. The relations between data should be expressed through relational metadata as specified in Deliverable 2.1, but a unified format for storing reconstructed and modeled virtual representations, which is also capable of maintaining accuracy information, is highly desirable.

Storage formats for 3D information

We start our discussion with an overview of up-to-date formats that are read and written by of-the-shelf software used in the post-production industry.

Name	**Ref**	**Description**
Alembic	[Alembic]	Alembic is a format for storing information about animated scenes after programmatic elements have been applied. It is discussed in the section

		Alembic.
DAE	[DAE]	COLLADA is an XML schema by the Khronos group for the interchange of 3D assets for industrial use. It has been standardized as ISO/PAS 17506 and is meant for the exchange of modeled, not reconstructed 3D scenes. DAE can describe cameras, but it cannot maintain individual points in a point cloud. Points must always be expressed as vertices of a mesh. It cannot express inaccuracy, either.
VTK	[VTK]	VTK (Visualization ToolKit) is focused on providing a format for storing multi-dimensional data that is then available to a variety of tools that can present and manipulate it. It has been tuned for the presentation of scientific data, where phenomena are rendering in a variety of ways to explore information contained in them. It allows the association of arbitrary data with points or cells in a dataset, including normals, matrices and indices. VTK has built-in abilities to encapsulate point clouds with convex hulls. Cameras in VTK terminology are only used for rendering, original data sources are not intrinsically understood, but an associated can be created through lookup in arbitrary tables.
FBX	[FBX]	Autodesk promotes FBX (Filmbox) as a 3D asset exchange format that facilitates high-fidelity data exchange their own and third party software. It has lost early version understanding of vertices, so it is not intended for storing point clouds. However, a developer can still store them as generic nodes with arbitrary attributes that can fulfill the role. Like in DXF, FBX can associated information with layers. Camera information can be encoded, and there is a concept of "producer camera". The FBX format is binary format (although an ASCII format exists), but model extensions can be stored using IO plugins. All FBX objects including generic nodes can be associated with arbitrary user-defined properties, allowing the introduction of accuracy.
LWO	[LWO]	Lightwave3D's LWO2 format is quite similar in scope to Alembic. It contains spatial, temporal, and structural information for rendering, but neither information about camera's that may have been used in initial reconstruction, nor accuracy information.
PCD_Vx	[PCD]	ASCII format for points clouds introduced by PCL, the point cloud library. It is currently at version 0.7. It can currently encode positions, colors, surface normals and moment invariants, as well as a viewpoint translation and rotation.
X3D	[X3D]	X3D version 3 is a development of the Web3D consortium and standardized by ISO/IEC. It is a continuation of VRML work. Version 4 is currently work in progress. X3D goes far beyond point cloud storage, including JavaScript scripting, animation, web integration and more. It can be encoded in several ways, including the SRC format, X3DOM, and the Efficient Binary Encoding (EBE). As a modeling standard, it has no standardized way of associating source cameras of a reconstruction to points. However, the X3DOM version is highly extensible, allowing arbitrary metadata attached to every point, or set of points.
OBJ		OBJ is a very strict ASCII format for encoding vertices, points, faces and textures first introduced by Wavefront Technologies.
PLY	[PLY]	The Polygon File Format (or Stanford Triangle Format) has an ASCII

		representation and a binary representation. It is inspired by the OBJ format that allows the definition of arbitrary properties for every point. This allows an implementation to add arbitrary information to points including accuracy information, but not in any backward-compatible way. Camera information could be included in comments.
STL		STL (STereoLithography) is an ASCII or binary file format by 3D Systems that was developed for 3D printing. It expresses faces and is not very suitable for point clouds.
OFF		OFF (Object File Format) is an extremely simple ASCII format that can only store vertices and faces.
C3D	[C3D]	C3D (Coordinate 3D) is a binary format was developed for photogrammetry software. Every point in a C3D file can have a residual value to represent the relative accuracy of the point. It is an extremely flexible format that supports, for example, to interleave 3D reconstructions with recorded original samples. At the same time, its main use for recording biomechanical processes is visible in the features that have been added; camera parameters are not among them.
DXF	[DXF]	Autodesk uses DXF (Drawing Interchange File Format) for handling AutoCAD drawings. Although it is meant for the construction of 3D scenes, DXF is capable of storing tolerances for points per axis. It does not provide obvious ways of describing camera parameters.
3DS MAX		The 3DS (3D Studio) MAX format superseding the older .3ds format seems to be entirely undocumented, and thus of little interest for us.
SVG	[SVG]	SVG (Scalable Vector Graphics) is a standardization activity by W3C aimed at potentially animated storage of vector graphics. SVG can store points as DOMPoint object with (x,y,z,w) coordinates. Without association with another element, DOMPoints are invisible abstract objects with no further use.
IGES	[IGES]	IGES (Initial Graphics Exchange Specification) is a standard for encoding is a standardized format for modeling circuits, surfaces and solid models. It can encode 3D points and associated colors, but it is fairly complicated to add further attributes.
SKP	[SKP]	Google SketchUp includes an API for manipulating files of the SKP type. The API allows encoding of 3D points, and their association with attributes from a user-defined attribute dictionary support arbitrary types. This makes it highly flexible although it does not come with many of the features relevant for LADIO.

Table 1: 3D file formats

The file formats listed in Table 1 are all in use in relevant tools for 3D construction or reconstruction. They provide quite a large number of features, but none of them is complete for recording all information that is interesting for LADIO. Obviously, we discount formats that are not suited for efficient storage of point clouds, and due to the goal of LADIO to support post-production, also formats that are incapable of formulating temporal changes of models are not suitable.

There are formats that are suited maintaining camera information, either for the purpose of maintaining information about the recording cameras that were used for a particular 3D reconstruction or for the purpose of rendering according to a camera specification. We know that changes of such camera parameters over time can be expressed externally, using the EBUCore extensions that we discuss in Deliverable 2.1. However, in the CMPMVS software that is at the heart of work packages 3 and 4, we subgroups consisting of two or more available recording cameras are involved in the reconstruction of every point and, in fast, the final selection of this camera group is made for every 3D point separately. Since this contributes to the understanding of accuracy mentioned at the beginning of this section, we would prefer a storage format that allows a direction association between pixels in a 3D reconstruction and the generating cameras.

The consequence of this ambition is that no existing format can support LADIO's requirement out-of-the-box using other functions than through generic attribution processes that we must implement into our software tools. Not all formats support generic attributes in this manner. From most obvious candidates from Table 1 are Alembic, COLLADA and VTK. FBX is less relevant because it has currently no sensible support for storing point clouds (interestingly, it has lost this ability from earlier versions). The main use case of FBX remains its role as a character animation interchange format, whereas Alembic bakes all animations frame by frame.

Of these formats, Alembic is the preferred choice for intermediate storage of points clouds for the LADIO partners working on this, because it is the only format that is already supported by software packages that are highly important in LADIO, including OpenMVG [AOM], Autodesk Maya [Maya], The Foundry's Nuke [Nuke] and Lightwave [LW3D].

However, in subsequent stages, i.e. after the integration of virtual assets into a scene and the creation of meshed structure from the point clouds, FBX becomes a contender. Although it is a format that is still under heavy development and not entirely stable, it is used by other relevant software such as assimp (Open Asset import library) [assimp], The Foundry's Modo [Modo] and Blender [Blender].

Alembic

Alembic [Alembic] is an open computer graphics interchange framework by LucasArts and Imageworks. It provides a way of generating geometric results for animated scenes, which is explained as the 3D equivalent of lighting and rendering scenes into 2D images.

The central task of Alembic is to store the result of this process efficiently, leading to a data representation for entire scenes, and the possibility of exchanging between artists. Examples of information that is stored as the output of Alembic are: enveloping, corrective shapes, volume-preserving simulations, cloth and flesh simulations. LADIO's preferred storage format for Alembic is Ogawa, replacing the earlier HDF5 [HDF5] that is still the default storage format of Alembic.

Since Alembic is highly expandable through plug-ins for providing input and export functionality, the limits of its ambition are best described by explaining what it is not.

Alembic does focus on the efficient storage of dependency graphs that are used as input for the procedural generation of geometric scenes. It does not store the pipelines of generating tools, and it does not maintain the rigs that lead to the final, animated scene, only the animated scenes that are generated from animating the rigs. It is an interchange format, rather than a final format native for any platform, and it is not an asset management platform. It is also inappropriate to take output stored in Alembic format as input for another geometry generation step, as each processing stage is probably not lossless.

As an interchange framework, Alembic is very flexible in providing software developers with the means of storing data in self-contained archives, and a huge amount of flexibility with respect to the information that they want to store.

For example, while it is unusual (although technically possible) to import raw input images that are used in the creation of a reconstructed point cloud, the locators for these files can be stored as a group of scalar properties (since string properties can be considered intrinsic). In software used at MIK, the original frames are stored separated, referenced from within the Alembic file. LADIO considers an approach that maintains the authoritative information in the metadata as described in Deliverable 2.1. There, the link between input frames is formulated as a ReconstructionJob connecting original Flows to an output VirtualScene. For compatibility with existing tools, this information will still be included into Alembic files. Textures are treated in the same way at MIK, although Alembic does provide a special Material module, which can in principle be used to bind textures to a surface.

In point clouds, all relevant information about points in a point cloud or cameras used for a reconstruction, can be added by means of adding properties to a child group. This is done with type descriptors, RGB color descriptors, and other information. In case of LADIO, this well-tested approach is used to add information that is currently not maintained, including accuracy information and a reference to the cameras that contributed to the reconstruction of a point.

Storage formats of Alembic

Alembic uses two underlying libraries to store data on disk. HDF5 is the original format that is understood by all tools that use any version of Alembic. Ogawa is included in the Alembic source code. It is a younger library, and older versions of some tools do not support it yet.

HDF5

HDF5 [HDF5] is a library for efficient storage and IO that is, among other things, used by Alembic. It has been created by Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratory (SNL) and Los Alamos National Laboratory (LANL). Among its field of use are visualization, statistics, and data analysis software, both open source and proprietary.

HDF5 promises a versatile data model that can represent very complex data objects and a wide variety of metadata, a completely portable file format supporting arbitrary number of data objects and unlimited data sizes, language bindings for C, C++, Fortran 90, and Java. Among its major advantages is the integration of high-performance features that optimize towards optimized access time and space consumption.

It provides a set of functions takes some inspiration from databases, such as the idea of accessing data by key, iterating through items, or using chunked tables that can be extended when additional data needs to be stored. It is also taking inspiration from file systems, including the idea of virtual mount points and mounting several file systems over each other; however, this is implemented below a concept of files.

HDF5 was previously the preferred storage option for Alembic. However, HDF5 has now been replaced by Ogawa. The announcements claims that Ogawa leads to speed-ups between 4 and 25 times of HDF5 speeds, and a file size reduction of 5 to 15%.

Ogawa

Ogawa is the result of a Google Code project initiated by Alembic. It knows the concepts of Data, Group, Archive and Stream.

Ogawa Data can be an item of arbitrary size that forms an intrinsic unit within a data structure. The Ogawa library does not interpret this data, but it should be created by a single add operation targeted at a group. This data can be overwritten, but never extended. Ogawa Groups represent a hierarchical structure between groups of Data, which allows both parent-child relationship and sibling relationships. Data can be created within a group, either in the shape of a single byte stream, or as a table of byte streams. It is possible to create data in this manner by calling the create member function of a group object, but not add the new data to this group; this has the implication that data can be stored in the file layout implied by a group, but be associated with several groups. It is also straightforward to create subgroups of a group. The interface allows the creation of child groups, as well as attaching existing groups as children. An Ogawa Archive is dedicated to storage of one group on disk. These archive may use an arbitrary number of Streams to perform actual read and write operations, but it provides a single interface for accessing a group. An Ogawa Stream is an abstraction for storing a byte stream on disk. It is used by the archive to read and write part of a group.

Based on this raw foundation, CoreOgawa adds concepts that distinguish between data and metadata, where metadata can hold arbitrary properties. Even the datatypes of these properties are defined by the user, and properties may take the format of scalar values, arrays or even compound properties. These properties may belong to a simple object or a compound object, which is always implemented and addressed as an Ogawa group.

There is furthermore the concept of sampling a group, and this allows an application to access the content of a group by sampling either in space or in time, where time is a subgrouping concept that differs from spatial attribute just by its special interpretation. The retrieval of a temporal sample provides access to a group with the closest time code to a requested time.

The result of this abstraction is that Ogawa archives are not self-describing, but they do carry sufficient information to allow application the retrieval of highly complex, non-homogeneous data structures.

Discussion of HDF5 alternatives

HDF5 is a very widely distributed storage format that is used in everything from scientific computing to film animation, and Ogawa follows is concepts very closely, but reduced complexity and effort in exchange for a focus on the restricted need for dimensionality in animation and film processing.

However, there are discussions about the feasibility of the HDF5 approach in general. Obviously, HDF5 attempts to provide efficient implementation also on data managements aspects that are classically expected from either operating system's file systems or from databases. The central argument for doing this is that the ability to manage data as a single file provides such huge benefits that is outweighs the efforts that lies in replicating operating systems' data management strategies in a library and requiring developers to understand appropriate chunk sizes.

Consequently, some developers argue in favour of a more classical approach, where the hierarchy of groups in HDF5 are replaced with a directory hierarchy and storage of data and its properties in flat files inside these directories. These discussions are led on the web, and examples are the articles by Cyrille

Rossant[1] and "Alex I"[2]. These flat files must obviously be structured, perhaps self-describing. A combination of an XML Schema and a compressed implementation should be capable of providing the same level of information. Alternatively, JSON files or simple CSV files have been proposed.

An important criticism from the HDF5 proponents is that files in the file system scenario require processing before they can be moved consistently to another computer. This is an important argument, and counters the proposal to use archiving utilities like zip and tar/gz for the task of data exchange. There are alternatives, like the disk images in MacOSX (.dmg) and Linux (loopback mounts). These disk images use operating system functions to achieve operating-system-specific good solutions for flat data block structures and provide encryption and compression, and support growth along with the dataset. Unfortunately, the sparse disk images that allow virtual size reservations and dynamic growth are not portable at this time, meaning that this option loses to the HDF5-like chunking operation.

Another proposed alternative is to store the information in a database, which may be an SQL database of any kind or a NoSQL database[3], including graph databases and including such with a JSON interface. The challenge with databases compared to HDF5 storage is that they tend to require fairly much administrative support to deliver high performance. HDF5 as used in an Alembic application used a limited number of data types, which can in principle be stored efficiently, but performing optimization requires administrators' involvement. Small database solutions that do not require central administration are of course also available, but those do not achieve the same read and write throughput as an application using the HDF5 library.

Accuracy maps

Generating and maintaining information about the accuracy position data is an important topic in LADIO, because it is required for several decisions between the film set and post-production steps leading to a final product. Several computations that extract information from recordings provide estimates of relative spatial and temporal positions that can be improved by applying them iteratively. In other cases, accuracy estimates indicate whether additional input data must be collected.

For example, the accuracy in 3D reconstruction of point clouds from images images is restricted by the accurate relative of two or more cameras whose pixels are used for estimating point positions. However, with decreasing angle between these cameras, uncertainty of the depth estimate increases, whereas increasing depth of the point with respect to the camera space implies that the image pixels covers a large area, increasing inaccuracy of the X-Y-position in camera space.

Whether such inaccuracy estimates are too large for a production cannot be expressed in absolute values, but depends on the requirements of the specific production. For example, real-time tracking of the position of the main camera on a film set may depend on feature extraction and matching of points with a point cloud reconstruction performed in advance. If the main camera's resolution of an essential part of a scene is much higher than the one that was used for reconstructing the scene, this leads to inaccurate tracking. Knowing this inaccuracy can either be used by improving reconstruction before the shoot, or

[1] http://cyrille.rossant.net/moving-away-hdf5/

[2]https://datascience.stackexchange.com/questions/262/hierarchical-data-format-what-are-the-advantages-compared-to-alternative-format

[3] A very long list of NoSQL databases is provided here: http://nosql-database.org/

even doing this in retrospect, and reconstructing again with higher accuracy for the relevant part of the scene.

Requirements in LADIO

LADIO relies on OpenMVG for point cloud reconstruction and CMPMVS for dense reconstruction of larger scenes from individual images. Neither OpenMVG nor CMPMVS do currently have the ability to store or export any accuracy information that may exist in intermediate stages of the reconstruction. From their reconstructions, we receive as output (sparse or dense) point clouds, where every point position is potentially estimated from a different pair of cameras (and validated with additional cameras). However, the link between these points and the cameras is broken, and the accuracy information cannot be extracted from this.

If the cameras used in estimating every point were known, we could use the relative camera pose in combination with the covariance matrix that collects the error information of the cameras' relative poses as it is generated by bundle adjustment from a pair of images. This covariance matrix maintains accuracy information for every parameter in the camera matrix after estimating its relative pose from images. However, there are other aspects that influence the accuracy of reprojected points, exceeding the inaccuracy of the camera pose itself; an example of such an additional influence is the radial distortion of the input image. Deliverable 3.5 has demonstrated how automatically deriving lens parameters has a tendency to concentrate good matches in image centers, and additional errors may be retained in points reconstructed from the outer pixels of a distorted image.

A thorough discussion about the error properties of bundle adjustment has been written by Triggs, McLauchlan, Hartley and Fitzgibbon [TMHF1999].

Relevant information

Work package 3 (Advanced 3D reconstruction) is in its early discussion phases. It is therefore not concluded what kind of data LADIO must finally store to account for accuracy information that is valuable for further processing with LADIO applications. At this point, it is reasonable to discuss the opportunities that lie in storing information in excess of that which is currently stored.

Information of accuracy is firstly available together with the cameras whose relative pose we estimate with respect to another camera recording the same scene. This can be expressed as a covariance matrix associated with the camera itself, and be stored either in the camera parameters in the metadata structures, or in the frame data that is used for the actual reconstruction. In LADIO, we store this covariance matrix with the reconstructed data, based on the understanding that reconstruction of camera pose and the scene itself is only possible together. Storing accuracy information about the camera pose would not have any benefit for selecting a stream that has been recorded by a camera, and create an particularly large amount of metadata when several cameras interact to identify the position of a particular 3D point.

The 3D reconstruction of a scene from a set of camera can generally be seen as the estimation of parameters representing both the camera parameters and every single reconstructed point. Whatever the technique used to propagate the uncertainty from measurements to parameters, the output is a covariance matrix for all parameters, which represents their uncertainty and dependencies. In principle then, it is conceivable to express uncertainty of all these parameters at once, and represent it in a single covariance matrix covering all cameras and points together. This covariance matrix would have thousands to millions of parameters.

It is not at all easy to compute uncertainty for large scenes (>100 cameras and 10000 points) in this way. Ceres solver, for instance, has only a generic uncertainty computation via general inversion by SVD (Singular Value Decomposition), which does not work on large scenes. More advanced methods are available but none really seem to be working for very large scenes. One of the problems in developing such methods is in creating ground truth, although CTU has produced ground truth for small scenes (50 cams, 5000 points) in Maple using higher precision computation (100 digits). It is thus an interesting research question whether an arbitrary-precision approach could be implemented, efficiently making use of GPUs, and this question will be raised in work package 3. For cases that are realistic today, however, using a complete covariance matrix must still be considered impossible, and it is actually often unnecessary. Instead, we will consider every point independently from all other points as well as the cameras, and every camera independent from all other cameras and points. With this independence assumption, we can represent uncertainty by symmetric covariance matrices.

Use in resectioning

In incremental reconstruction, cameras are incrementally added to a reconstruction step, by estimating their camera parameters by matching image correspondences with cameras that are also present as references. This estimation is camera resectioning.

Maintaining accuracy information for all camera poses would help to estimate the uncertainty in pose estimations to decide which cameras should be attached to building the model, meaning that parameters estimated from cameras whose own pose was determined with higher accuracy should be given a higher value in estimating new cameras' poses. The information could also provide input to a heuristically chosen cut-off that decides to remove or avoid cameras whose own pose is not well-determined, losing very little information in resectioning. This could help in the reconstructions process by allowing to make an iterative selection of cameras to add, which provide the best view in terms of their own pose estimate's accuracy. Selecting a sequence of cameras to add in reconstruction would implicitly maintain a high spatial accuracy of reconstructed points as well.

Use in localization

Image-based camera localization in LADIO works by finding feature correspondences between a recording camera's (main camera or witness camera) frames and frames that are stored in a database of images that was used for 3D reconstruction of a scene.

For the purpose of estimating the camera pose, we search for the pose in a static scene. In contrast to the use in resectioning, the location of the new frame in space, as well as the camera parameters, are only based on retrieval of the relation between earlier frames. Even though bundle adjustment is performed during estimations of the new camera pose, and outlying point in the original reconstruction are detected, this tracking operation performs only read operations on the database. Thus, outliers will never be removed or rated down. If there are wrong points, those will not be modified by the new camera, and they may even have a strong impact on random views of the tracked camera.

Therefore, it would be desirable to maintain uncertainty on points of the original reconstruction. Not only points could be excluded, or guidance based on the points' accuracy could be integrated artificially into bundle adjustment. Bundle adjustment can actually use the covariance matrix of every pose that led to the reconstructed point to weight the point in the cost function of the bundle adjustment that is used for tracking.

This method can be used by storing a covariance matrix for every point in the database (general choice), it can be used for an arbitrary view direction based on the stored covariance matrix, or it can be done along the principal axis of the camera only.

Storing accuracy as attributes

It is sensible to consider every reconstructed 3D point independent from all other points and cameras, and all cameras independent from other cameras and all points. Perhaps at some stage of SfM computation, one might still consider dependencies between cameras and points for selecting points and cameras in incremental/global SfM but that is not yet clear. For an n-parameter entity, one has to store n*(n+1)/2 parameters of, e.g., upper triangular part of the matrix and its diagonal. This means that we must store 6 parameters for points and a larger number of parameters for perspective cameras.

For different assumptions and camera representations that were discussed in the context of this deliverable, representations leading to covariance matrices with 13, 23, 66 and 78 parameters were proposed. The decision about the appropriate representation is part of the work package 3 effort, it must here suffice to say that the representation of camera parameters as a Scalar Property belonging to a camera representation in Alembic is capable of satisfying all of these options.

Keeping the symmetric matrix is feasible in all cases: the covariance matrices can be implemented as Scalar Property objects (which can be a constant-sized array) that can be attached to every point as well as to every camera. This is the same approach that is taken to specify the color or normal of a point. In terms of precision, it should be sufficient to store single precision floating point values for all parameters.

However, depending on experience that LADIO gains during work package 3, it may turn out that it is feasible to represent the accuracy of points in a simpler manner. For example, LADIO's choice may be to store only 3 semi-axes of the uncertainty ellipsoid, i.e. eigenvalues of the covariance matrices, or just one maximal semi-axis to express an uncertainty sphere, or even a 2-parameter uncertainty cone, to save storage space. This simplified approach should work relatively well for points only, doing the same for the camera is less obvious due to its dimensionality. For visualization, it could still be useful to express the camera's optical center in this manner.

Filesystems, Databases and other Data Collections

The discussions about HDF5 and its alternatives have shown that it is not obvious that complex information should be stored in a proprietary file structure. This approach forces the software developer to understand both the features of the application that he supports very well, guessing the size of the appropriate blocks ("chunks" in case of HDF5) that is appropriate for potentially expanding tables of content, but the developer must also be aware of the underlying operating system and the constraints of the particular system hardware that is used by the computer where his software is probably deployed. Again, the segmentation (chunking) plays an essential role in keeping performance high in loading and mapping data between computers' main memory and storage.

Classically, these decisions have been separated into the concerns of the application developer and the operating system (OS) developer, where the overall structure and the concrete block layout become the problem of the OS developer, while the application developer cares primarily about application needs. It can be feasible to map structures onto other storage methods.

File systems

The primary storage system is computers is its set of file systems. File systems have a very long history, starting with the so-called CMS minidisks and later, FAT of DOS. These file systems did hardly deserve the name, since they provided only flat collections of badly named files, where some characters of the name were reserved to indicate the file type.

The modern equivalents of these file systems feature a hierarchical namespace (a tree) with a clear distinction between the inner nodes (called catalogue, directory or folder) and leaf nodes (files and special files). It is generally possible to simulate directed acyclic graphs of inner nodes by means of special files that redirect a leaf node to an inner node or a leaf node elsewhere in the tree.

This structure is quite capable of emulating the group structure that is created by Alembic. The main counter-argument is that deeply structured Alembic file can be transferred as a unit, whereas files in a file system require an additional step to create an easily transferrable file from the hierarchical structure.

Furthermore, since Alembic is accessed through a software layer, it is capable of locking a structured unit as a whole, allowing only threads to modify any part of the group structure under single, central control. This ability is inherent to the Alembic interface, whereas any similar ability in a file system would, first, require explicit locking of all files involved using a separate operating system mechanism (e.g. lockd), second, it would be open to unintentional manipulation by other tools. On the second point, however, it is important to note that any corruption of an Alembic file invalidates all data in the structured unit, whereas only a subset of files may be affected in a file system.

Modern general purpose file systems

Microsoft's New Technology File System (NTFS) is the primary file system for Windows. It can be used on large partitions and hold files or arbitrary size. Like all modern file systems, it supports access control, file locking and encryption. Each file is stored as a file descriptor in a Master File Table (MFT), separate from the file content. The MFT is replicated once and contains all metadata of the file. NTFS understands the concept of sparse files and can store them efficiently.

Microsoft's Resilient File System (ReFS) is a new development that is preferred for servers since Windows 8. It accounts for the large number of files that must be maintained on a server and structures the entire file system in B+ tree. The central MFT is replaced by a decentralized and much more resilient structure, that is also faster to access for large numbers of small files.

The most frequently used file system on Linux systems today is Ext4. It adds to the older version of the file system that concept of extents [RO1992], which does not only save blocks that were required for data management in earlier versions, but also provides a speedup since more data of larger files can be read contiguously. This is combined with reservation, as well as aggressive write caching, both to increase the size of such extents. It is aware of stripe sizes to make use of multi-disk partitions. Files can be up to 2^{44} bytes in size.

SGI's file system XFS was ported to Linux [MEL+2000] due to a need for better handling of a workload mix of very large contiguous files, sparse files, large numbers of small files, deep directory hierarchies and directories with large numbers of files. It should also be resilient. XFS features a B+-tree indices on all data structures, caching of content and all data structures, and support for parallel processing. For resilience, it uses a journaling approach. Files and file systems can be up to 8 TB large. In contrast to the original SGI version, the Linux version is not capable of outright reserving resources CPU and IO resources for guaranteed performance, relying on the efficiency of a cheaper best-effort implementation. In spite of its age, it is still preferred over Ext4 for handling media files.

IBM's JFS was ported from AIX [BGH2003]. Whereas XFS was developed with multimedia workloads in mind, JFS was meant for servers holding a huge number of small files, intense parallel access, and running on top of several layers of disk virtualization. These layers do provide interesting features, however, such as reducing average data access times by storing "hot" files in central tracks of a disk, while moving "cold" files in the same file system to the edge tracks. Interestingly, the final feature set is very similar to XFS. Journaling is part of the name, but data structures are arranged in B+-trees, individual extents of a file can be up to 2^{24} blocks in size, where the block size is chosen at file system creation time. The virtualization features of JFS allow efficient parallel processing. To deal with the hard choice between large blocks for large files and small blocks to avoid the overhead for large numbers of tiny files, JFS gained the ability to store multiple files in a single block. JFS retains a replicated superblock to store information about the filesystem. Due to its origins as a media file system, XFS is more likely to be found in the media processing context.

BTRFS [RBM2013] is a newly implemented file systems for Linux that inherits many ideas from XFS and JFS. It's special abilities include snapshotting of the entire file system for backup purposes, which is a feature that can be implemented in a fairly straight-forward manner on top of a journaling file system. It features also compression (a feature dropped from JFS in an earlier generation) and efficient handling of a devices mix consisting of spinning disks and SSDs. It has the ability to store mapped memory using copy-on-write, which is especially interesting in the presence of a large number of temporary files.

ZFS [Pow2012] was developed for Solaris, but it now also an interesting file system for Linux users and was temporarily important on MacOSX. ZFS has dynamic block sizes and supports data pipelining. Like in JFS, disks must first be pooled by means of a volume manager, before ZFS can use them. If provides striping, copy-on-write and snapshots like BTRFS.

Apple has implemented a variety of features that resemble BTRFS in HFS+.

Special purpose file systems

Special purpose file systems were quite popular for both multimedia and the high-performance community. Multimedia file systems focussed on an ability to support the best possible number of parallel isochronous data streams requested by several users concurrently from spinning disks, while high-performance file systems focussed on the storage of large amounts of multi-dimensional data with a high raw single-stream throughput, which could be accessed efficiently in arbitrary slices. The many interesting ideas that were derived from these file systems have been integrated into today's mainstream file systems.

The new general purpose file systems are related to data processing. Specifically HadoopFS has been created with a single kind of operation in mind, perform map-reduce operations on huge datasets. To achieve this, HadoopFS builds on large blocks that would be highly inefficient in general-purpose applications but beneficial for the particular case.

Google FS (GFS) [GGL2003] is nearly a general purpose distributed file system, but due to the large amount of data that is hosted by Google, its primary purpose is replicated storage to overcome the unavoidable defects of disks and computers at large scales. At the same time, GFS should have a performance similar to a local file system. It has been observed that it trades an increase in access latency for a good data throughput. As such, it is quite suitable for processing of tasks similar to HadoopFS.

Databases

As explained before, Alembic is composed of hierarchical data tree that can depend on or be depended on by other hierarchical data trees. Given that the way to use data stored in Alembic is to load data in bulk from disk into memory for local processing, we have no reason for using the interactive functions of databases in LADIO applications. Consequently, it is not reasonable to use a database as main storage for any LADIO application.

That being said, there may be specific cases where the use of a database could prove efficient. For such cases, the Alembic structure allows that the chosen technology depends on the practical need. More precisely, if the need relates to quickly walking through dependencies, a Graph database should be favoured, whereas if the need is related to bulk loading of data for a processing or conversion operation, a document-store or a relational database (which is more close to the Alembic structure and easily convertible to an Alembic structure) should be favored. Finally, it is worth noting that a document store would allow to easily export Alembic trees in Python dictionaries, Python being a language that is broadly used in the industry, both by tool vendors and LADIO implementers and users.

There is another aspect of database that is interesting for LADIO. While low-end database system tend to exist on top of a file system, high-end database systems tend to allocate their own memory on a pool of disks. These are then managed completely by the database system, foregoing the file system logic. The reason for this is that databases are (still) mostly maintaining data and relations in tabular form, and both read and write accesses manipulate these tables. Since very small and local entries in database tables change much more frequently than other stored content, it is not reasonable to write entire new blocks of data as a journaled file system would. Rather than that, databases update data blocks in sizes corresponding to memory pages, and split and shrink the block share that is used for a section of a table, and write them back to disk at once.

This is a feature that is highly desirable for the kind of content that LADIO is manipulating. Alembic does currently not require this functionality because it is assumed that the computer main memory (potentially with large amounts of swap space) can deal with the situation of dynamic blocks and the developer defines a reasonable chunk size. Since Alembic is designed for data growth rather than data change, this is mostly feasible. However, if reconstruction in LADIO grows to sizes that exceed reasonable main memory assumptions, or our systems don’t use unified memory any more, page-wise read and write operations similar to those of database systems are a very promising step, which is also elaborated in Storage arrangements.

Object stores

Several were several approaches to unify the abilities of filesystems and databases, under the assumption that one memory management paradigm could supersede the other. The attempts came from both sides, operating systems and database researchers, and they always made the assumption that one's own paradigm could solve all problems. In real-world deployments, the integration has so far not succeeded. However, out of concepts such as the ObjectStore database systems [LLOW1991] came the current approach of understanding content in terms of typed objects that can have a multitude of requirements. Among the current topics in object store research are the conflict between consistency and aggressive world-wide replication [NRT2016] and, more relevantly for LADIO, the efficient placement of and access to very high quality media involving the entire system stack including routers [MMK+2017].

While these research challenges are only partially solved, LADIO focusses on practical applications that can mostly rely on the state of the art. Here, document stores like Elastic [Elastic] and Apache CouchDB [CouchDB] have already been used by the partners. Both provide a REST API and query operations formulated in JSON, which provides a straight-forward integration with the LADIO data models developed in deliverable D2.1. Obviously, both of these object stores are meant for storage of partially or fully unstructured data, and both understand the concept of binary blobs. This implies that it would be possible to store media content as discussed in the earlier sections of this document in these object stores. In the context of LADIO, this makes no sense, since our media data is mostly not textual and indexing media content makes no sense. Consequently, these object stores are useful to us for their API, which is well-suited for our metadata, including both the structured metadata discussed in D2.1, and any annotations that are provided during the production process.

Storage arrangements

With the emergence of SSDs and other flash memory-based storage devices, traditional IO bus technologies, such as SATA and SAS/SCSI, have become inadequate. Compared to mechanical spinning disks, SSDs have considerably higher speeds. Consequently, the maximum throughput of SATA has become a bottleneck, and SSD manufacturers now use the PCIe bus instead.

As older spinning disks are primarily limited by disk rotation and seek time, the older Advanced Host Controller Interface (AHCI) [AHCI] did not reflect the low latency and parallelism modern multi-CPU architectures are capable of. NVM Express (NVMe) [NVME2017] was designed from ground-up to fully exploit the functionalities of the PCIe bus and reflect the designs of parallel systems.

The key difference between NVMe and AHCI is the concept of command queues, and where the queues are hosted. Traditional AHCI controllers have on-board RAM memory where command queues are hosted. A CPU needs to do multiple writes over the IO bus in order to prepare disk commands, which the disk controller then executes. As multiple CPUs may to the the same on-board memory, they need a synchronisation lock to coordinate their efforts. Upon command completion, the disk controller will issue a single interrupt, requiring a predetermined CPU to determine which command has completed and which userspace process is affected.

In contrast to AHCI, where the CPU is required to write to the disk controller's on-board memory in order to set up disk commands, NVMe aims to reduce the number of IO bus accesses a CPU must perform by relying on the coordination already existing in PCIe instead. The specification allows command queues to be hosted in system memory, rather than on the disk itself. When commands are ready to be submitted, the CPU will do a single write to a dedicated register associated with the queue, reducing the IO bus accesses to a single write operation and in addition eliminating the need for any synchronisation between CPUs. The disk controller will then use its own DMA engine to fetch the command and execute it. Upon completion, the disk controller notifies the host system of completions by posting them to a completion queue associated with the individual command queues. The NVMe specification also specifies that a disk controller can use dedicated interrupts per command queue.

As disk commands are prepared in the host system memory, a NVMe disk controller has an understanding of memory pages. This resonates well with file system that operate on page levels and support memory-mapped files and to disk block-aligned data structures, where a disk controller can read from or write to file blocks and user-space buffers, eliminating the need for intermediate kernel-level buffers. By hosting commands in system memory, this also allows a file system to perform certain optimisations where the data access pattern is largely known in advance. Commands can be prepared in different memory pages (and in different queues if necessary) in advance, and a CPU can simply swap addresses before triggering a command fetch, resulting in extremely low latency disk IO.

In addition, as NVMe disks reside on the same IO bus as other PCIe devices, the disk controller is able to read from and write to any PCIe device that support RDMA, for example certain GPUs. With an RDMA-capable GPU and a NVMe compliant SSD, it is possible to transfer data between the GPU and the SSD without involving the CPU at all.

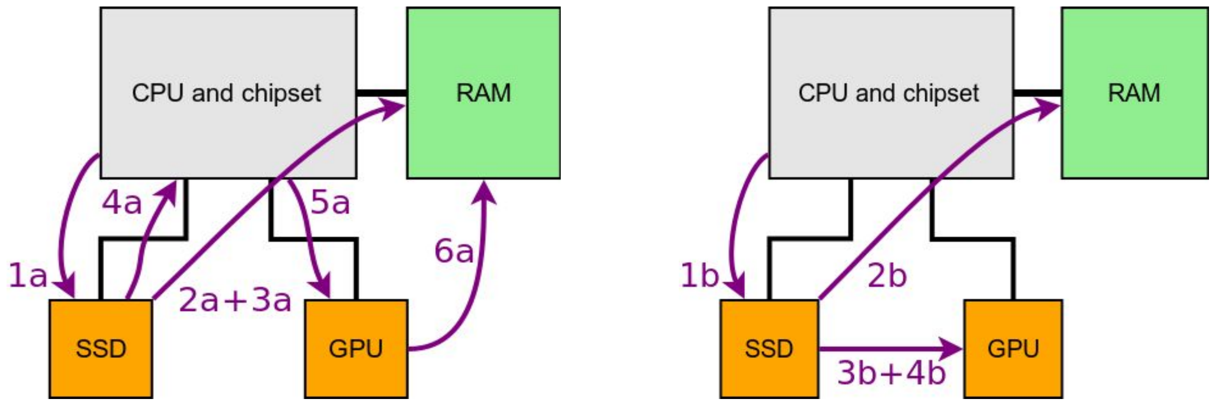

Figure 6: Traditional disk IO vs direct disk IO

Figure 6 illustrates how data transfer can be optimised this way. The CPU triggers a command fetch (1a), which the disk controller then fetches and executes (2a). In the traditional method, the disk would then write blocks to system memory (3a) and notify the host using an interrupt (4a). The host would then tell the GPU to read data from system memory (5a), which the GPU would then do (6a). Using direct disk IO, however, steps 1b and 2b are the same as 1a and 2a. The difference is that the disk would write directly into GPU memory (3b) and then notify the GPU by posting a completion to GPU memory (4b), thus significantly reducing latency and freeing up CPU-time.

PG-Strom [Kag2016] is an extension to PostgreSQL that exploits the benefit of having a GPU and an SSD reside on the same PCIe fabric. The extension hooks into the Ext4 file system and make the disk and the GPU appear to the database application as a storage unit with SQL offloading mechanisms. Under the hood, data is directed directly into GPU memory, and the GPU then performs simple SQL queries before the data is passed system memory and accessible to the database application. The benefit is that datasets that are larger than what can fit in system memory can be preprocessed and reduced in size before being accessed.

CMPMVS reconstructs larger scenes from individual images, by comparing any one image frame with six other reference frames. As GPUs have limited memory, it is unlikely that all frames can be held in active GPU memory simultaneously. While the process of copying frames between GPU and system memory can be pipelined, it requires synchronisation points between CPU and GPU where one is idle waiting for the other to complete. It may even be the case that the working dataset is larger than what can fit in system memory, adding the complexity of having to synchronise disk IO as well.

A workaround is to control disk access from the GPU itself, by hosting a user-space NVMe driver on the GPU [Mar2017]. Using a graph-like or tree-like database to determine dependencies in advance, it is trivial to prepare disk commands in advance and allow the GPU to point an SSD to the relevant IO commands on demand, eliminating the need for synchronisation between GPU and CPU. This would most likely require a conversion process that is executed in advance, unpacking image frames from encoded and compressed files into a block- and memory page-aligned format suitable for direct IO.

References

[TMHF1999] Triggs, B., McLauchlan, P. F., Hartley, R. I., & Fitzgibbon, A. W. (2000). Bundle Adjustment - A Modern Synthesis. In Proceedings of the International Workshop on Vision Algorithms: Theory and Practice (pp. 298–372). London, UK, UK: Springer-Verlag. http://dl.acm.org/citation.cfm?id=646271.685629

[Gre1986] Greene, N. (1986). Environment Mapping and Other Applications of World Projections. IEEE Computer Graphics and Applications, 6(11), 21–29. http://doi.org/10.1109/MCG.1986.276658

[VF1994] Voorhies, D., & Foran, J. (1994). Reflection vector shading hardware. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94 (pp. 163–166). New York, New York, USA: ACM Press. http://doi.org/10.1145/192161.192193

[Paw2011] Pawson, R. (2011). OpenGL ES 3.0 Programming Guide, Second Edition. Evaluation (Vol. 17). http://doi.org/10.1177/1356389011400889

[GGL2003] Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03 (p. 29). New York, New York, USA: ACM Press. http://doi.org/10.1145/945445.945450

[AHCI] Serial ATA Advanced Host Controller Interface (AHCI), Revision 1.3.1. Intel Corporation.

[NVME2017] NVM Express, Revision 1.3 (2017). http://www.nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf

[Kag2016] KaiGai, K. (2016), GpuScan + SSD-to-GPU Direct DMA http://kaigai.hatenablog.com/entry/2016/09/08/003556

[Mar2017] Markussen, J. (2017). CUNVME - A userspace NVMe driver implemented in CUDA https://github.com/enfiskutensykkel/ssd-gpu-dma

[Ed2015] Graham Edwards (2015). How Lidar is Used in Visual Effects. Reprinted on tested.com, URL http://www.tested.com/art/538855-how-lidar-used-visual-effects/ (last accessed 31. May 2017)

[Se2012] Mike Seymour (2012). Pointcloud9 – a LIDAR case study. URL https://www.fxguide.com/featured/pointcloud9-a-lidar-case-study/ (last accessed 31. May 2017)

[ASPRS2013] ASPRS (2013). LAS Specification Version 1.4 – R13 15 July 2013. URL http://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf (last accessed 31. May 2017)

[ISO2005] ISO/IEC (2006). ISO/IEC 14496-10:2005 Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding. URL https://www.iso.org/standard/43058.html (last accessed 31. May 2017)

[ISO2015] ISO/IEC (2015]. ISO/IEC 23008-2:2015 Information technology -- High efficiency coding and media delivery in heterogeneous environments -- Part 2: High efficiency video coding. URL https://www.iso.org/standard/67660.html (last accessed 31. May 2017)

[Apple2017] Apple (2017). Apple ProRes White Paper. URL https://www.apple.com/final-cut-pro/docs/Apple_ProRes_White_Paper.pdf (last accessed 31. May 2017)

[Avid2012] Avid (2012). Avid DNxHD Technology. URL https://www.avid.com/static/resources/US/documents/dnxhd.pdf (last accessed 31. May 2017)

[RO1992] Rosenblum, M., & Ousterhout, J. K. (1992). The design and implementation of a log-structured file system. ACM Transactions on Computer Systems, 10(1), 26–52. http://doi.org/10.1145/146941.146943

[MEL+2000] J. Mostek, B. Earl, S. Levine, S. Lord, R. Cattelan, K. McDonell, T. Kline, B. Gaffey, R. Ananthanarayanan (2000). Porting the SGI XFS File System to Linux. Proceedings of the Annual Conference on USENIX Annual Technical Conference, http://dl.acm.org/citation.cfm?id=1267724.1267758

[BGH2003] S. Best, D. Gordon, I. Haddad (2003). Kernel korner: Ibm's journaled filesystem. Linux Journal (105). URL http://www.ee.ryerson.ca/~courses/coe518/LinuxJournal/elj2003-105-jfs.pdf

[RBM2013] O. Rodeh, J. Bacik, C. Mason (2013). BTRFS. *ACM Transactions on Storage*, *9*(3), 1–32. http://doi.org/10.1145/2501620.2501623

[Pow2012] H. Powell, Howard (2012). ZFS and Btrfs: A Quick Introduction to Modern Filesystems. Linux Journal (218). URL http://dl.acm.org/citation.cfm?id=2328941.2328946

[OSC2017] Google (2017). Open Spherical Camera API Specification. API level 2. URL https://developers.google.com/streetview/open-spherical-camera/

[LLOW1991] C. Lamb, G. Landis, J. Orenstein, D. Weinreb (1991). The ObjectStore database system. *Communications of the ACM*, *34*(10), 50–63. http://doi.org/10.1145/125223.125244

[NRT2016] J. Neto, V. Rancurel, V. Tao (2016). Managing Object Versioning in Geo-Distributed Object Storage Systems. In *Proceedings of the ACM 7th Workshop on Scientific Cloud Computing - ScienceCloud '16* (pp. 11–18). New York, New York, USA: ACM Press. http://doi.org/10.1145/2913712.2913714

[MMK+2017] M. Montagnuolo, A. Messina, E.K. Kolodner, D. Chen, E. Rom, K. Meth, P. & Ta-Shma (2016). Supporting media workflows on an advanced cloud object store platform. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16* (pp. 384–389). New York, New York, USA: ACM Press. http://doi.org/10.1145/2851613.2851620

[Flea3] PointGrey Fleat3 USB3 camera. URL https://www.ptgrey.com/flea3-32-mp-mono-usb3-vision-sony-imx036-camera

[M2S120M] DO3Think M2S120M USB2 camera. URL http://www.dsicam.com/products/USB2_CAM/

[HERO] GoPro HERO cameras. URL https://gopro.com/help/productmanuals

[Phan3] DJI Phantom 3 4K Drone. URL http://www.dji.com/phantom3-4k/info

[Phan4] DJI Phantom 4 Drone. URL https://www.dji.com/phantom-4/info

|Parrot] Parrot Bebop Drone. URL http://global.parrot.com/au/products/bebop-drone/

[Liblas] LAS 1.0/1.1/1.2 ASPRS LiDAR data translation toolset. URL http://www.liblas.org

[LASlib] Martin Isenburg. LAStools: converting, filtering, viewing, processing, and compressing LIDAR data. URL http://www.cs.unc.edu/~isenburg/

[PUCK] Velodyne LiDAR. PUCK VLP-16. URL http://velodynelidar.com/vlp-16.html

[ZEB1] GeoSlam. ZEB1. URL http://www.geoslam.com/hardware-products/zeb1

[Vu8] LeddarTech. LeddarVu Solid-State LiDAR Platform. URL http://leddartech.com/modules/leddarvu/

[Osram] Osram Opto Semiconductors. Meilenstein für Laser-Sensoren in selbstfahrenden Autos. URL http://www.osram-group.de/de-DE/media/press-releases/pr-2016/07-11-2016

[Alembic] Alembic. Homepage. URL http://www.alembic.io/

[DAE] M. Barnes, E.L. Finch (2008). COLLADA – Digital Asset Schema Release 1.5.0. URL https://www.khronos.org/files/collada_spec_1_5.pdf

[VTK] Kitware Inc. VTK User's Guide. URL http://www.vtk.org/vtk-users-guide/

[FBX] AutoDesk. FBX SDK Programmer's Guide. URL http://docs.autodesk.com/FBX/2014/ENU/FBX-SDK-Documentation/index.html

[LWO] Lightwave (2001). LWO2 file format for 3D objects. URL http://static.lightwave3d.com/sdk/2015/html/filefmts/lwo2.html

[PCD] PointCloud Library. The PCD (Point Cloud Data) file format. URL http://pointclouds.org/documentation/tutorials/pcd_file_format.php

[X3D] Web 3D Consortium. X3D Version 4. URL http://www.web3d.org/x3d4

[PLY] PLY Files: an ASCII Polygon Format. URL https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html

[C3D] C3D.ORG. C3D file format. URL https://www.c3d.org/HTML/default.htm

[DXF] AutoDesk. DXF Reference. URL https://www.autodesk.com/techpubs/autocad/acadr14/dxf/dxf_reference.htm

[SVG] W3C (2017). Scalable Vector Graphics (SVG) 2. Editor's Draft 09. URL https://svgwg.org/svg2-draft/

[IGES] E. Reid (2001). The Initial Graphics Exchange Specification (IGES) Version 6.0. URL https://filemonger.com/specs/igs/devdept.com/version6.pdf

[SKP] Google. SketchUp C API. URL http://extensions.sketchup.com/developer_center/sketchup_c_api/sketchup/index.html

[HDF5] The HDF Group. HDF5 Software Documentation. URL https://support.hdfgroup.org/HDF5/

[AOM] Alicevision. Alicevision OpenMVG branch. URL https://github.com/alicevision/openMVG

[Maya] AutoDesk. Maya. URL https://www.autodesk.com/products/maya/overview

[Nuke] The Foundry. Nuke. URL https://www.foundry.com/products/nuke

[LW3D] LightWave3D. LightWave. URL https://www.lightwave3d.com

[assimp] Assimp Open Assert Import Library. URL http://assimp.sourceforge.net

[Modo] The Foundry. Modo. URL https://www.foundry.com/products/modo

[Blender] The Blender Foundation. Blender. URL https://www.blender.org/

[Elastic] Elastic. ElasticSearch. URL https://www.elastic.co/

[CouchDB] Apache. CouchDB. URL http://couchdb.apache.org/