Heritage Building Information Modelling (HBIM) Course

Annada Prasad Behera
School of Computer Sciences, NISER
Bhubaneswar, OD 752050
annada.behera@niser.ac.in

Subhankar Mishra
School of Computer Sciences, NISER
Bhubaneswar, OD 752050
smishra@niser.ac.in

Abstract

Divided into two modules, this handout for the Heritage Building Information Modelling (HBIM) course illustrates the creation of digital twins and HBIM for historic architectural sites which uses the Rajarani Temple in Bhubaneswar as an example. The first module goes into the process of data collection with help of photogrammetric data collected with unmanned aerial vehicles and terrestrial laser scanners and then the creation of the digital twins, with best practices for sampling data for superior results. The module also goes into rendering of the digital twins created. In the second module, the storage and creation of the database is discussed, with illustrative examples of database scheme. Course link : https://www.niser.ac.in/∼smishra/project/hbim/hbimcourse/

Module I: Data Collection

1 Introduction

This module is about the data collection, where we will discuss about how we collect the data, process the data and store them. We will do a brief overview of some tools (both proprietary and open source) that will help in the process. The type of data collected is dependent on the HBIM dimensions.

For the first three dimension of the data, which are the three spatial dimension for which we will need to survey the actual location and build 3D model out of it. The two most common techniques are terrestrial laser scanning, TLS and photogrammetry using unmanned aerial vehicle UAV.

The other dimensions require data collection from other sources, such as organizations like Archaeological Survey of India, private parties and stakeholders in the subject cultural heritage. We go brief into such data collection.

At the end, we will discuss how the above was applied while collecting the data for Rajarani HBIM. In summary, the module will cover the following:

- Terrestrial Laser Scanning (TLS)
- Photogrammetry using unmanned aerial vehicle (UAV)
- Other data collection sources
- Proprietary and open source software for data processing
- Case study of the above for Rajarani
- Legal, technical and non-technical challenges for the above mentioned parts of this module

2 Photogrammetry

Photogrammetry is the process of obtaining reliable information about physical objects and the environment through processes of recording, measuring and interpreting photographic images and patterns of recorded radiant electromagnetic energy and other phenomena.

Though photogrammetry has been as old as photographs itself, the techniques have gone from mostly analog and optical techniques to computer-aided simulations. For heritage sites, the photogrammetry is a very helpful technique to estimate the geometry. In general, the photogrammetry can be divided into two types, (a) aerial and (b) terrestrial based on the position of the camera and also based on the position of the target, it can be divided into long and short range photogrammetry.

For applications that require high image/geometry details in conjunction with occluded regions like the rooftops, terrestrial photogrammetry is not enough and hence the close range aerial photogrammetry is preferred. For such applications, we need to operate a unmanned aerial vehicle (UAV) such as a drone. In the course further, UAV and photogrammetry will be used interchangeably.

The basic principle behind photogrammetry is the geometrical reconstruction of the path of rays from the target to the camera sensor at the moment of exposure.

2.1 Photogrammetry Pipeline

In this section, we will describe how a set of images are converted to a 3D model and go through the various steps involved, such that we can have a clear idea on how we can tweak the parameters and other factors so that the process of data collection and processing can be optimized for better final model. The goal is not to go into the detail mathematics behind the process, nor the details of the algorithm or its implementation.

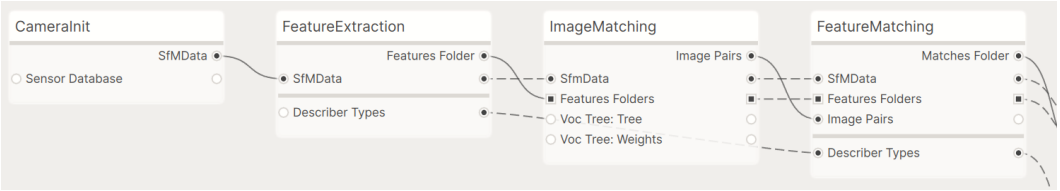

Figure 1: Photogrammetry pipeline illustration.

2.2 Natural Feature Extraction

The natural feature extraction is to find a group of pixels that are not changing too much for different viewpoints. One of the well known method for doing so is called Scale-Invariant Feature Transform or the SIFT algorithm.

For a given image, a set of progressively down-scaled images are first created and then points with locally maximum Laplacian are identified. These maxima are the points of interest or features. The Laplacian is a measure of how the color value of the current position is different than the values at its neighboring points. The square patch of pixels with the center at the maxima is called key-points.

The number of patches are sometimes reduced to a reasonable number if they are too high. Each extracted discriminative patch is then stored along with its descriptor based on the gradients around the maxima.

SIFT extracts discriminative patches from the first image and compares them to the discriminative patches in the second image, irrespective of the rotation, translation or scaling. These patches and their transformation gives clues for the changing of camera viewpoints during photographing.

2.3 Matching

In this part of the pipeline, the aim is to find the areas in the scene that are similar in different images. That can be done with the help of descriptors of key-points. By converting the entire image into a compact image descriptor, we can compare two images or parts of it by simply computing the norm

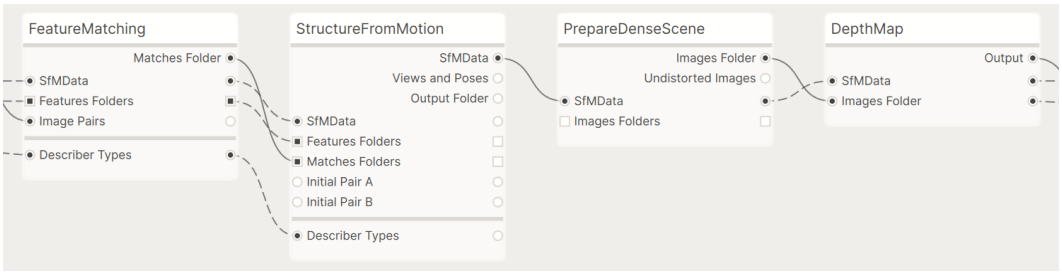

Figure 2: Feature extraction and matching steps.

between them instead of comparing every feature individually that may take a lot of computation power.

The features are then all matched between candidate image pairs, based on their image descriptors.

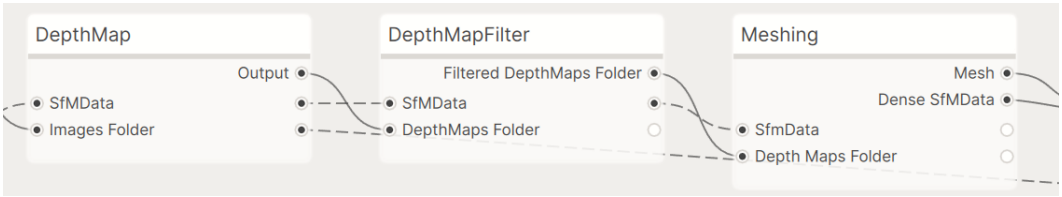

Figure 3: Image matching stage.

2.4 Structure from Motion and Depth Map

The Structure from Motion algorithm calculates the camera positions of all the images. This is a very popular algorithm and is extensively documented and we won't go into details of it. Once the camera positions are computed, the depth value of each pixel can be calculated using many approaches such as Block Matching, Semi-Global Matching, ADCensus, etc. For the depth map of independent images (which can be computed in parallel), a filtering step is applied to ensure consistency between multiple cameras.

2.5 Meshing

The structure from motion and depth maps computed is used to create a dense geometrical representation from the scene. The depth map is used to create a trochee of the depth values and then the process of tetrahedralization is performed to get an interconnected mesh. The mesh is then separated using a min-cut max-flow algorithm.

A Laplacian is applied to remove rogue points in the mesh and unnecessary vertices.

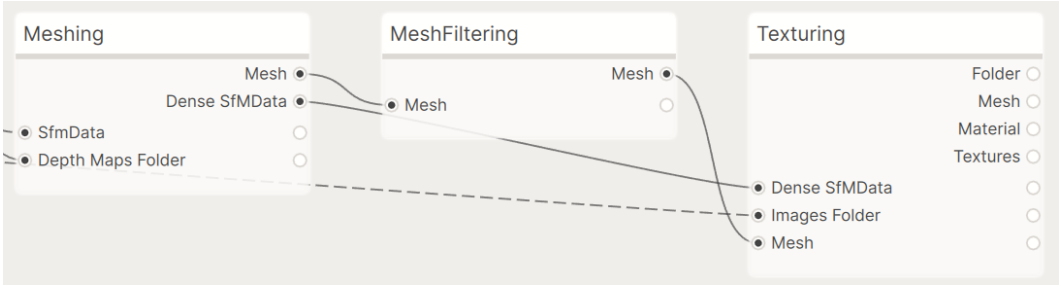

Figure 4: Meshing process.

2.6 Texturing

The generated mesh generally doesn't have a UV associated with it. The UVs are computed automatically into regions of triangles based on the original triangulation of the mesh.

For every triangle, many candidate images are considered that are visible from the given camera angle such that they are looking at the surface perpendicularly. The color is then averaged over each triangle and stored in the texture map.

We have glossed along the pipeline for the entire photogrammetry process. A basic overview of the process can be helpful in case we want to fine tune our photography to produce better models.

3 Photographing Techniques

In this chapter, we explore how to produce 3D polygon meshes from photographs. The quality of the model depends on various factors, and understanding how each affects the final output helps in generating higher-quality reconstructions.

The initial step in the photogrammetric pipeline is feature detection, which builds a point cloud of the model. Thus, a good photograph maximizes the number of detectable features on the subject while minimizing background features. All adjustments listed below aim to enhance this feature detection.

Although automatic settings on digital cameras suffice for everyday photography, they fall short for photogrammetric reconstruction. This section outlines important camera settings: shutter speed, aperture, and ISO—collectively known as the exposure triangle.

1. **Camera Shutter Speed** Shutter speed defines the duration the camera sensor is exposed to light. A longer exposure (lower shutter speed) results in brighter images but risks blurriness if either the camera or subject moves. Blurry images reduce feature detection quality and degrade reconstruction accuracy.

   To avoid motion blur, use a higher shutter speed. While this results in darker images, it minimizes sensor exposure time, leading to sharper images. Higher shutter speed is ideal for well-lit subjects, but low-light conditions pose challenges.

   In such scenarios, stabilize the camera using a tripod and a remote shutter release to prevent movement. For drone photography, ensure the drone is stationary while capturing images.
2. **Camera Aperture** The aperture is the size of the opening in the lens, controlled by the f-number (e.g., f/2, f/32). A larger aperture (lower f-number) lets in more light and results in a shallow depth of field, which blurs the background. A smaller aperture (higher f-number) reduces light intake but increases the depth of field.

   For cultural heritage documentation, a shallow depth of field is often beneficial, as it suppresses distracting background details and emphasizes the subject. Background features may be erroneously detected and tracked, wasting computational resources.
3. **Camera ISO** ISO controls the sensor's sensitivity to light. A lower ISO value improves dynamic range and color accuracy while reducing chromatic aberrations and image noise. Low-noise images improve feature detection reliability. However, low ISO requires higher light exposure, making it essential to balance ISO with shutter speed and aperture settings.
4. **Subject Zoom and Overlap** The photogrammetric algorithm does not require the full subject in every shot. Close-up images that emphasize detectable features are preferred. Ensure at least 70% overlap of features between adjacent images to support effective triangulation.
5. **Subject Material** Highly reflective or transparent surfaces are problematic, as reflected features may be misinterpreted. Similarly, fine structures like hair or nets are difficult to reconstruct. In such cases, manual modeling is often more effective.
6. **Subject Masking** Complete isolation of cultural heritage sites from their background is rarely possible. If depth of field alone doesn't sufficiently blur the background, digital masking is necessary to eliminate background features that consume processing power.
7. **Number of Views** To capture complete models, take photographs from diverse angles and elevations, including occluded regions. At least three images per region are necessary for accurate triangulation. While post-processing can fill gaps, accuracy may suffer.

3.1 Summary

In summary, consider the following guidelines:

- Use higher shutter speed, lower ISO, and low depth of field.
- Capture close-ups with abundant detectable features.
- Ensure 60–70% feature overlap between images.
- Avoid reflective, specular, and transparent materials.
- Avoid blurry images.
- Avoid subjects with fine or loose parts like hair or nets.
- Mask the subject from the background where necessary.
- Ensure the subject occupies at least 70% of the frame.
- Use varied elevations to capture occluded and shadowed areas.
- Capture more images than necessary—more data helps, provided computing resources are available.

With these guidelines, the data collection process can begin, setting the stage for the next phase: photogrammetric reconstruction.

4 Photogrammetry

The flight plan for the unmanned aerial vehicle is as follows:

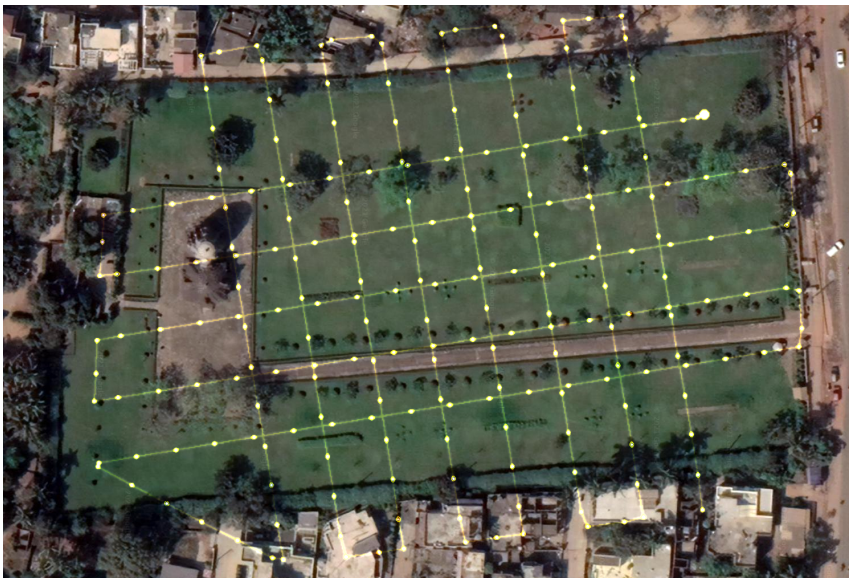

Figure 5: Drone trajectory.

The drone was flying at about 80 meters above the ground.

Photogrammetry relies on the overlap of images—the same features need to be detected in at least three images. The images are stitched through a process beginning with densification and key-point detection. Once enough overlapping images are acquired, they are calibrated and the camera positions are estimated.

The next step is to manually provide control points. The base-station used was an RTK-based GNSS system (real-time kinematic correction). The drone acted as the rover while the station acted as the base. The remote, rover, and base were in constant communication. This configuration enabled real-time geolocation calculation and correction of coordinates.

Figure 6: Base station.

The geolocation device was mounted on both the craft (rover) and the base station. Civilian GPS typically has a 5 to 8 meter error, which this system helps to correct. The system was connected to a CORS network (Continuously Operating Reference Station) and a VRS (Virtual Reference Station).

The sensor was a 1" CMOS 20 megapixel camera with a 24mm lens and 84° field of view. The UAV had automatic pitch-roll-yaw stabilization. The flight was conducted at an altitude of 80 meters, at a speed of 7 m/h, with 75

Control points were manually added to improve measurement accuracy. Occluded regions during the overall coarse scan were addressed by using a flat vertical scan profile with close-up images.

To prevent motion blur, the drone synchronized shutter speed with movement. Since photographs were taken in RAW format, parameters like gamma and color can be corrected in post-processing (except shutter speed). Tone, temperature, and color balance corrections were applied through batch processing, primarily for tonal balance. Gamma correction remains manual.

4.1 Rajarani Camera Details

Camera Parameters	**Value**
Number of photographs	2334
Shutter speed	120
ISO	100
Aperture	f/5.0
Focal length	24.0mm (35mm film), 8.8mm (lens)
Metering method	Average

Table 1: Camera specifications for Rajarani Photogrammetry

Figure 7: Terrestrial laser scanner.

5 Terrestrial Laser Scan

The scanner used was a FARO Focus M70, operating via time-of-flight laser pulses at a scan speed of 97 Hz, capable of capturing approximately half a million points per second, within a range of 1 to 30 meters and with 3 mm accuracy.

A total of 62 scan positions were selected, based on the intricacy of the carvings and the goal of minimizing scan errors. A built-in 360° 165 megapixel HDR camera with 2x exposure bracketing was used to capture the color and texture of the point cloud. Each vantage point required about 13 minutes.

The scanner was positioned strategically to minimize coverage error. Note that it does not include GPS capability; geo-tagging is done post-collection via software. Some areas inaccessible to the scanner were filled in manually using photogrammetry data.

Figure 8: Scanning with an obstruction with ASI installed grill on the ceiling.

Blind spots included hard-to-reach corners and the temple's upper regions. After data collection, overlapping scan points were registered manually by identifying common features and triangulating points to determine absolute coordinates.

The temple interior was dark; a torch was used inside the *garbha griha* and the corridor connecting it to the *jagamohana*. A metallic grill, installed by ASI, obstructed scans of the ceiling; the scanner was placed atop the grill to capture this region.

The ceiling area of the *garbha griha* had a large bat population, making scanning difficult. Gardeners were enlisted to clear the bats so technicians could safely operate. Above this ceiling lies the hollow interior of the *deula*, which was also infested with bats. A temple worker managed to capture a few images of this interior.

Figure 9: The hollow interior of the temple's *deula*.

6 Photographing Techniques

In this section, we explore how to produce 3D polygon meshes from photographs. The quality of the model depends on various factors, and understanding how each affects the final output helps in generating higher-quality reconstructions.

The initial step in the photogrammetric pipeline is feature detection, which builds a point cloud of the model. Thus, a good photograph maximizes the number of detectable features on the subject while minimizing background features. All adjustments listed below aim to enhance this feature detection.

Although automatic settings on digital cameras suffice for everyday photography, they fall short for photogrammetric reconstruction. This section outlines important camera settings: shutter speed, aperture, and ISO—collectively known as the exposure triangle.

1. **Camera Shutter Speed** Shutter speed defines the duration the camera sensor is exposed to light. A longer exposure (lower shutter speed) results in brighter images but risks blurriness if either the camera or subject moves. Blurry images reduce feature detection quality and degrade reconstruction accuracy.

   To avoid motion blur, use a higher shutter speed. While this results in darker images, it minimizes sensor exposure time, leading to sharper images. Higher shutter speed is ideal for well-lit subjects, but low-light conditions pose challenges.

   In such scenarios, stabilize the camera using a tripod and a remote shutter release to prevent movement. For drone photography, ensure the drone is stationary while capturing images.

2. **Camera Aperture** The aperture is the size of the opening in the lens, controlled by the f-number (e.g., f/2, f/32). A larger aperture (lower f-number) lets in more light and results in a shallow depth of field, which blurs the background. A smaller aperture (higher f-number) reduces light intake but increases the depth of field.

   For cultural heritage documentation, a shallow depth of field is often beneficial, as it suppresses distracting background details and emphasizes the subject. Background features may be erroneously detected and tracked, wasting computational resources.

3. **Camera ISO** ISO controls the sensor's sensitivity to light. A lower ISO value improves dynamic range and color accuracy while reducing chromatic aberrations and image noise. Low-noise images improve feature detection reliability. However, low ISO requires higher light exposure, making it essential to balance ISO with shutter speed and aperture settings.
4. **Subject Zoom and Overlap** The photogrammetric algorithm does not require the full subject in every shot. Close-up images that emphasize detectable features are preferred. Ensure at least 70
5. **Subject Material** Highly reflective or transparent surfaces are problematic, as reflected features may be misinterpreted. Similarly, fine structures like hair or nets are difficult to reconstruct. In such cases, manual modeling is often more effective.
6. **Subject Masking** Complete isolation of cultural heritage sites from their background is rarely possible. If depth of field alone doesn't sufficiently blur the background, digital masking is necessary to eliminate background features that consume processing power.
7. **Number of Views** To capture complete models, take photographs from diverse angles and elevations, including occluded regions. At least three images per region are necessary for accurate triangulation. While post-processing can fill gaps, accuracy may suffer.

6.1 Summary

In summary, consider the following guidelines:

- Use higher shutter speed, lower ISO, and low depth of field.
- Capture close-ups with abundant detectable features.
- Ensure 60–70
- Avoid reflective, specular, and transparent materials.
- Avoid blurry images.
- Avoid subjects with fine or loose parts like hair or nets.
- Mask the subject from the background where necessary.
- Ensure the subject occupies at least 70
- Use varied elevations to capture occluded and shadowed areas.
- Capture more images than necessary—more data helps, provided computing resources are available.

With these guidelines, the data collection process can begin, setting the stage for the next phase: photogrammetric reconstruction.

7 Rendering

In this section, we discuss lighting, materials, and rendering the final output mesh. Numerous tools, both open source and proprietary, exist for rendering a mesh. These include standalone renderers like POV-Ray and Mitsuba, as well as fully integrated 3D software suites such as Blender and Autodesk Maya. In this section, we use Blender, as it is free, open source, and available on most major operating systems including BSD and Haiku.

7.1 Materials

Blender offers an extensive node-based system for shading 3D objects with materials, textures, and associated manipulations. In this section, we focus on manipulating these to achieve desired rendering results.

Texture maps reconstructed from photographs provide information on how to color each polygonal face in the mesh using UV coordinates. While generally accurate, these may require tuning or re-generation if results are unsatisfactory, particularly with shiny or reflective surfaces.

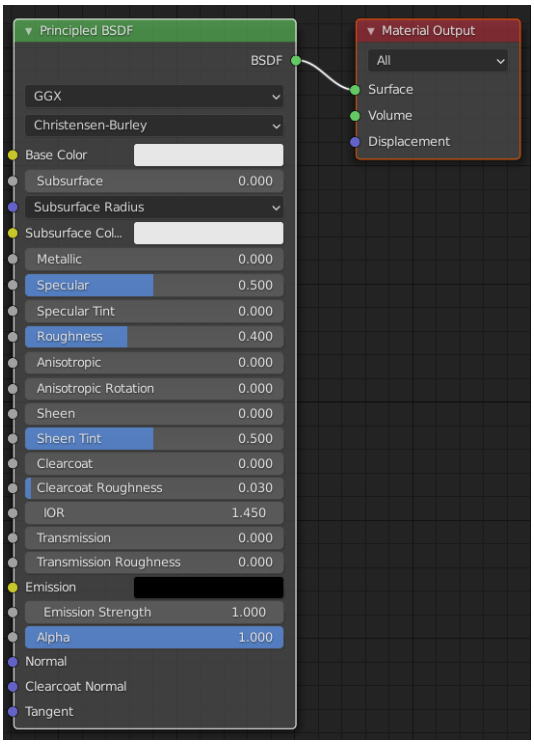

Figure 10: PBR materials node in Blender.

For example, when a surface and its texture are shiny, light reflection during path tracing must be simulated. Blender, like many other 3D software tools, uses a Principled BSDF shader, often referred to as the PBR (Physically-Based Rendering) node.

Key parameters in the PBR shader include:

- **Base Color**: Controls the object's base coloration. Can be uniform or driven by a texture with UV mapping.
- **Subsurface Color**: Defines the color of subsurface light scattering. Rarely applicable to rocks and masonry but can be relevant for translucent materials.
- **Metallic**: Dictates reflectivity; higher values simulate metals.
- **Specular**: Controls intensity of direct reflections.
- **Roughness**: Affects diffusion of reflected light; high values produce matte finishes.
- **IOR (Index of Refraction)**: Used for transparent objects to simulate light bending.
- **Transmission**: Controls the degree of transparency.
- **Normal**: A normal map input, with RGB values corresponding to the surface's orientation.

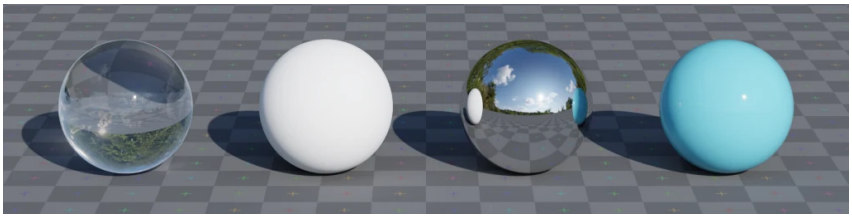

Figure 11: PBR materials.

Material behavior varies with parameter values. For example:

- Transmission = 1.0 yields a fully transparent material.
- Roughness = 1.0 creates a fully matte surface.
- Metallic = 1.0 simulates a highly reflective metal.
- Roughness = 0.0 and Metallic = 0.0 gives a highly polished, non-metal surface.

7.2 Lighting

Real-world scenes are illuminated by diffuse natural and artificial light. In a digital scene, the absence of light results in dark renders. Conversely, excessive lighting can wash out the scene.

A balanced solution is to use an HDRI (High Dynamic Range Image) or environment map. When light rays in path tracing do not intersect any object, they take the color of the environment map. Many free environment maps are available online, and 360-degree photos can also be used in Blender.

HDRI lighting provides a realistic ambient light setup, illuminating even shadowed or occluded regions as seen in real-world settings. However, if environmental lighting is unsuitable, a three-point lighting setup can be used.

The three-point light setup includes:

- **Key Light**: The main light source that defines the scene's illumination.
- **Fill Light**: A weaker light that reduces harsh shadows.
- **Back Light**: Positioned behind the subject to separate it from the background and highlight the silhouette.

Blender includes a built-in addon for setting up three-point lighting: `Lighting: Tri-lighting`. Activate it from `Edit > Preferences > Addons`, then add it through the `Add` menu.

Lighting and materials affect render quality significantly, but so does the camera. The same principles discussed in the photographing techniques section apply in rendering. Blender supports the relevant camera parameters as well.

8 Rajarani HBIM Project

The data for Rajarani was collected in the year 2021 for the purpose of preservation and protection of the Rajarani temple through a digital twin in the form of Heritage Building Information Modeling (HBIM) and a semantic web of knowledge encompassing various HBIM dimensions.

The following data acquisition methods were employed:

- Terrestrial Laser Scanning
- Unmanned Aerial Vehicle (UAV) Photogrammetry

The collected data was processed into the following outputs:

- Highly detailed 3D polygon mesh with textures
- 3D point cloud
- High-resolution photographs of the temples
- Virtual reality (VR) experience compatible with Oculus headsets
- 2D ground and elevation architectural plans

All data was consolidated into an HBIM framework and a semantic web of knowledge using the techniques discussed throughout this course.

The complete technical paper for this project can be accessed at: *[link to be inserted here]*.

Module II: Database and HBIM

1 Introduction

In this module, we will assume that the data has already been collected and processed, and we will tackle the problem of creating a **semantic web of knowledge** such that the processes of adding, querying, updating, and viewing the data are more accessible.

In summary, the aim of this module is to discuss:

a) the advantages of using HBIM and a semantic web of knowledge,

b) how to build a semantic web of knowledge and HBIM as flexibly as possible, such that adding new data and features is as easy as possible,

c) various components of the proposed model, such as the API, database design, etc.,

d) incorporation of new machine learning models or algorithms into the already collected and processed data, and

e) data migration in the face of a potentially new design.

Along with this, we will also discuss the pitfalls and rationale for the choices we have made, and how we intend to approach potential problems with the current design.

2 Setting up the Interface

We will use the Python programming language to build the interface for the HBIM. Python can be downloaded from the official site, and installation instructions differ across operating systems. For Linux and macOS, the recommended method is using the default package manager.

Alternatively, independent package managers like Miniconda help install different versions of Python and many Python packages in a virtual environment without affecting the operating system's packages.

Since the database management software is PostgreSQL, we need to interface PostgreSQL from Python, for which we will need the `psycopg2` package from PyPI, which can be installed by issuing the command:

```
pip install psycopg2
```

3 Connecting the Database

Before we can connect to the database, we must ensure that the database server is up and running. The operating system's init system generally starts the server daemon, and its status can be checked by:

```
sv status postgres           # for other init systems
systemctl status postgres    # for systemd init system
```

If the service is not running, it can be started by switching to the `postgres` user and issuing the command `postgres`, or by adding it as a service to the init system.

Once the database is up and running, we can connect to it using `psycopg2`:

```
#!/bin/env python
import psycopg2 as pg

db = pg.connect(
    host='localhost',
    database='hbim',
    user='postgres'
)
```

Specific to the psycopg2 package, all database operations are executed as cursor commands. To test if the database is working, we will execute the "Hello, World!" program statement in SQL:

```
cur = db.cursor()
cur.execute('SELECT "Hello, World!";')
for row in cur:
    print(row)
```

Since a SELECT statement in SQL usually returns tuples of the relation, the cursor executes the query and stores it as a Python list iterator, which can be iterated over to get individual tuples. The tuple is a string constant:

```
('Hello, World!',)
```

This confirms that the database connection was successful. If you do not see the above result, please ensure that the database server/daemon is running and the `psycopg2.connect()` function exited successfully.

4 Images

The database contains the `images` relation, and it is recommended that all operations on images be performed through the interface. The interface must expose the following functions:

- Add new images to the database
- Add comments to the images
- Add placeholders to the images
- Add bookmarks to the images
- Remove an image from the database

The images are not stored as binary blobs in the database but as files in a central repository.

5 Extending the HBIM with New Data

The HBIM is designed for extensibility, allowing integration of new data and improved querying capabilities. This section demonstrates such an enhancement through two examples.

Example I: Document Transcripts

Motivation The HBIM database includes various documents, but they are not machine-readable, hindering search. Using Optical Character Recognition (OCR) tools like Tesseract can solve this.

Tesseract is an open-source OCR engine based on LSTM neural networks. Though its core is in C++, it has front-ends for many languages including Python. Interfacing with the database can be done independently of the API language.

Initial Database Schema The current structure of the `documents` relation is:

```
CREATE TABLE documents(
    id              SERIAL PRIMARY KEY,
    description     TEXT,
    filetype        CHAR(8),
    file            CHAR(20),
    comments        TEXT,
    top_left        POINT,
    bottom_right    POINT
);
```

However, this does not support storing OCR transcripts. To enable this, we define a new table:

```
CREATE TABLE transcripts (
    id       SERIAL PRIMARY KEY,
    texts    TEXT,
    doc_id   INT REFERENCES documents(id),
    page     INT
);
```

Here, `texts` holds the OCR output, `doc_id` links to the `documents` table, and `page` indicates the page number.

Tesseract Setup Tesseract can be installed using the system's package manager. For macOS:

```
brew install tesseract
```

Populating the Database Once Tesseract is configured, the OCR process can extract text page-wise from document files and populate the `transcripts` table.

5.1 Advantages

- Direct access to data minimizes overhead.
- Any language that can interface with the database can be used.

5.2 Disadvantages

- Requires knowledge of the database structure.
- Significant schema changes may necessitate API updates.

6 The Database

There are many robust database management systems available like MySQL, MariaDB, Oracle. But since we are only going to use open source tools, we will use PostgreSQL. PostgreSQL is based on relational database model and the create, read, update and delete (CRUD) operations are done using the Structured Query Language (SQL).

Getting the Database The official binary releases and source code can be found on at PostgreSQL, which should be the preferred method of installing the database software on Windows and Mac OS X. For Linux-based operating systems, the package manager should be preferred.

Postgres uses a client-server model, unlike say SQLite where the main program provides both the user interface and the database management. The server for Postgres is program called postgres. There are many clients for Postgres having both command-line and GUI interfaces for Windows, Mac, Linux as well as the web, with both proprietary and open-source license. In short, there are a lot of choices for the client, and the official Postgres Wiki has a list, which can be found here.

In this text, we will be using the Linux operating system.

Setting up the Database Server Once the installation is complete, the server can be invoked by simply issuing the command postgres. However, the server will most likely fail to start because it doesn't come pre-configured. The official Postgres documentation, which can be found here explains how to configure the Postgres server. We will not go into details, but for the sake of completeness we will provide a minimal working example.

The Postgres server must run as a different user, since the server can be exposed to the public and as a different user the postgres server has no access to the user files. Some installation of postgres already creates a new user while some of them do not. To create a new user we can issue the command,

```
useradd postgres -d /var/lib/postgresql
```

Listing 1: Creating a Postgres User

where the -d flag denotes the home directory of the postgres user.

The server stores the data in a directory and we have to create the data directory and init the database as the user postgres.

```
su - postgres
cd ~
mkdir data
initdb -D /var/lib/postgresql/data
```

Listing 2: Initializing the Database

For Ubuntu users, the recommended way of switching users is `sudo su postgres`. Please substitute if you are using Ubuntu. The server daemon can now start by issuing the following command (as the postgres user).

```
postgres -D /var/lib/postgresql/data
```

Listing 3: Starting the Postgres Server

For the HBIM, we will create a new database (again, as the user postgres user).

```
createdb hbim
```

Listing 4: Creating the HBIM Database

Setting up the Client Among the myriad of clients available, we will use the official `pgsql` client that is maintained by the Postgres developers. It is free, open source like the Postgres server and it is a command-line interface. The client can be invoked by,

```
psql
```

Listing 5: Invoking the pgsql Client

If you are inclined to use other client, please see their documentation.

7 Database Design

There are 7 dimensions of HBIM and we want to store all the data so that they can be easily accessed. While the API takes care of the data read and write operations, there are cases when direct access to the database is desired. For instance, if we want to run a algorithm to perform some task, say OCR, on all the scanned and collected documents, then accessing the database directly is faster and easier, since we do not have to worry about the API layer. However, the disadvantage of doing things this way is that it requires knowledge about the internal database design.

In this section, we will be going through the database design process for storing the various dimensions of the HBIM as well as the interplay of those dimensions that depend on each other. At the outset, we want the HBIM to support the following features:

- Support for adding comments and placeholder texts for images, videos, documents and models.
- Support for bookmarks at various points and time.

7.1 Images

As a primary key, we can add an `id` attribute to distinguish between tuples and to refer from other relations as a foreign key. Postgres supports storing the images as a binary blob in the database. But we want to avoid that. The most important reason for not storing the data as a binary blob is that the image cannot be accessible without the database. In case of a database corruption or database integrity issue, the images will be lost. To prevent this, we store the image in a separate directory and refer to it from the database. That attribute is a string called `file`.

We can add a description to the image as a string called the `description` attribute. With that, the entire relation for images would look like:

```
CREATE TABLE images (
        id                SERIAL PRIMARY KEY,
        description       TEXT,
        file              CHAR(20),
        comments          TEXT,
);
```

Listing 6: Image Table Creation

As you can see, the `file` name is limited to 20 characters including the file extension.

7.2 Comments

To add support for comments in images, we need to store images as a relation in the database. The relation database must contain the attribute for the comments as well as the location in the image in pixels. For example, we can add a comment, like "ruins" at the pixel position (100,100) or a region of space with the corners, (100,100) and (200,200) pixel position. The comment can be stored as a string `comment` attribute and the corners as `top_left` and `bottom_right` as points. Points are a predefined datatype in Postgres.

To store the comments, create the following relation:

```
CREATE TABLE comments (
        id             SERIAL PRIMARY KEY,
        description    TEXT,
        page           INT,
        top_left       POINT,
        bottom_right   POINT,
        type           ENUM,
        type_id        INT REFERENCES images(id) documents(id) video(id
    ),
        time_stamp     TIMESTAMP,
);
```

Listing 7: Comment Table Creation

The `page` and `time_stamp` attributes can be NULL. For documents, the comments are given a location but they can also be at different pages. The page table specifies at which page the comment is located. The `time_stamp` attribute does the same for videos. The `type_id` is the foreign key to respective relation.

7.3 Documents

The document can be stored same as the images relation with some extra attributes like the file type. Images, irrespective of their file types are, generally supported by image viewers. But for documents, there are different filetypes that need different viewers. For example, a PDF viewer may not support DJVU files. So, we store the file type in the database. With that, we can create the following relation:

```
CREATE TABLE documents(
        id                SERIAL PRIMARY KEY,
        description       TEXT,
        filetype          CHAR(8),
        file              CHAR(20),
        comments          TEXT,
);
```

Listing 8: Document Table Creation

Most of the filetypes have 3 characters extensions, but some have 4 or even 5 characters extensions. Therefore to accommodate it, and future filetypes, we have allocated 8 characters in the database.

8 Acknowlegement

The authors would like to acknowledge the support from the following organizations: **(a)** Mr. Arun Malik and Dr. Arunima Pati, Archaeological Survey of India, Bhubaneswar Circle for their help

throughout the data collection process, **(b)** Directorate General of Civil Aviation, Ministry of Civil Aviation for granting necessary permission; **(c)** ATC Tower, Bhubaneswar for smooth operation of drones; **(d)** Fibrox 3D solutions, Unmanned and Airborne Solutions Pvt. Ltd. and Sens Geomatic Pvt. Ltd. for drone and laser scanning equipment and operations.