Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Augmented Reality

Visual Coherence and 3D models

Manuela Chessa – manuela.chessa@unige.it
Fabio Solari – fabio.solari@unige.it

Summary

Visual Coherence

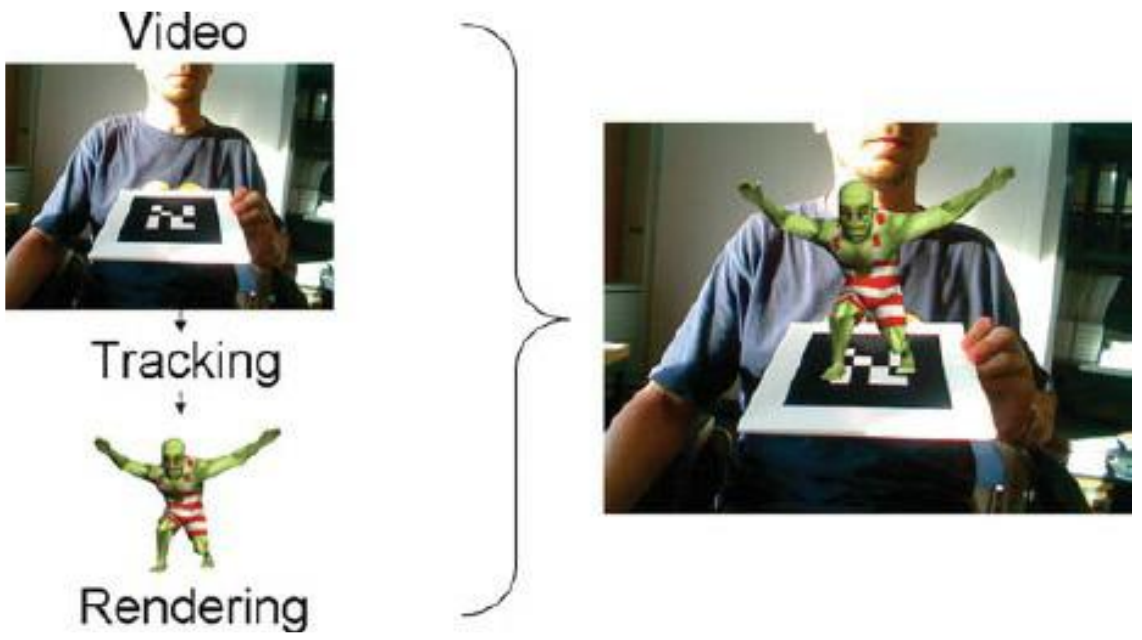
- Occlusion detection among real and virtual objects
- Ground plane detection
- Photometric registration

Visual Coherence

- We examine how to obtain **visually coherent output** from an AR system.
- We investigate how real and virtual objects can be combined so that the virtual objects blend seamlessly into the real environment.
- We have already discussed one important component of visual coherence, **spatial registration**.
- Here, we concentrate on **appearance**-related issues.

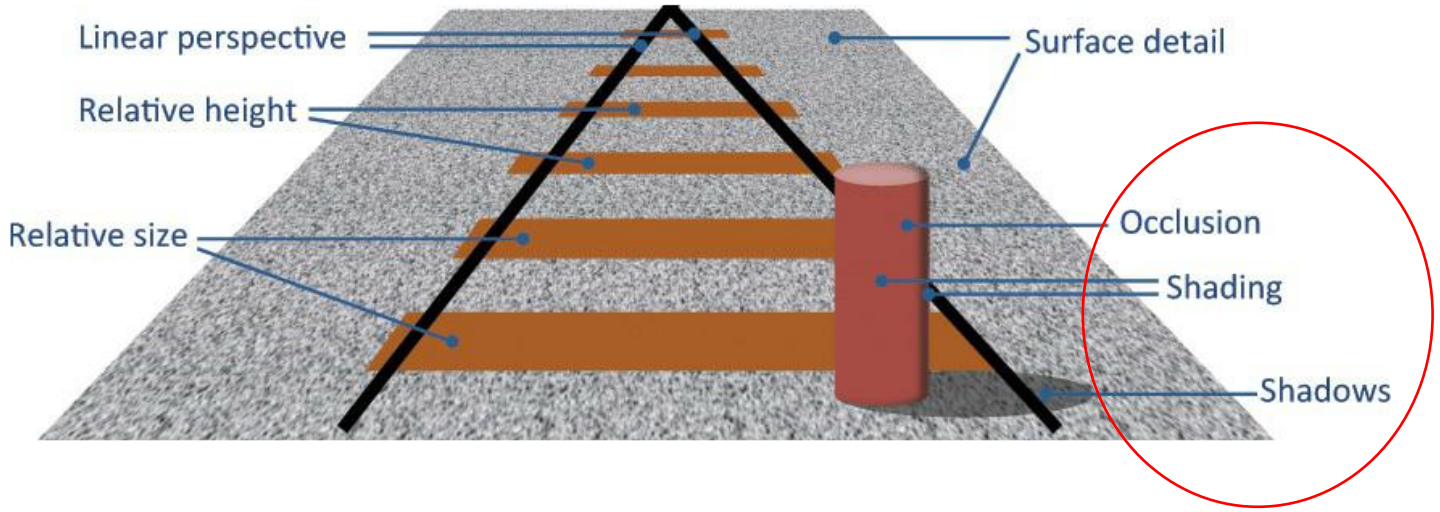
Visual Coherence: spatial registration

- Given the **relative pose** of the observer (or **camera**) to the **scene**, virtual objects can be positioned and oriented correctly in the output image.
- The virtual object must be rendered from a **virtual camera** with internal and external camera parameters corresponding to the **real camera**. With such a **calibrated camera**, essential **depth** cues can be generated.



Visual Coherence: depth cues

- There are approximately 15–20 different **depth cues**, which can be broadly categorized as monocular or binocular cues.
- **Binocular** cues can be produced only by a stereoscopic display device.
- Given that many of today's AR displays use a **monocular** video see-through mode, monocular depth cues are more **important for AR** in general. They can be produced purely by means of **computer graphics** software.



Visual Coherence: depth cues

- Three-dimensional computer graphics are well equipped to deliver all of these cues.
- However, **occlusion**, **shading** and **shadows** require special treatment in AR rendering.
- To bring together virtual and real, AR rendering must **extend** a conventional **computer graphics pipeline**: a pipeline for visual coherence must perform acquisition, registration, and compositing of multiple data sources.

Data Source	Acquisition	Registration	Compositing
Geometry	Geometric reconstruction, SLAM	Geometric registration	Occlusion, diminished reality
Light sources, materials	Light probes	Photometric registration	Common illumination
Images	Video capture	Camera calibration	Camera simulation, stylization

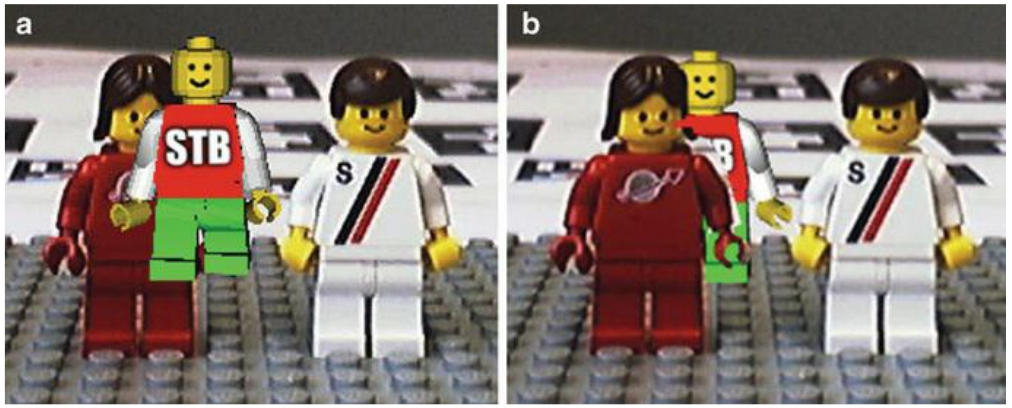
Visual Coherence: AR graphics pipeline

- We therefore assume a pipeline for video see-through AR, which consists of the following stages:
  1. **Acquisition**: acquire a model of the real scene (geometry, materials, illumination).
  2. **Registration**: Establish common geometric and photometric properties between the real scene and the virtual scene.
  3. **Compositing**: merge the real scene and the virtual scene into a single image.
  4. **Display**: present the composite image to the user.

Data Source	Acquisition	Registration	Compositing
Geometry	Geometric reconstruction, SLAM	Geometric registration	Occlusion, diminished reality
Light sources, materials	Light probes	Photometric registration	Common illumination
Images	Video capture	Camera calibration	Camera simulation, stylization

Occlusion handling in video-see through

- Simply drawing computer graphics objects on top of a video background with a registered camera is not sufficient to create the impression of a scene in which the *virtual and the real coexist*.



- (a) *The virtual character is placed at the correct position, but occlusion by the physical character is not considered*. (b) **Correct occlusion** rendering creates a much **more realistic** impression without conflicting cues.

Occlusion handling

- Occlusion is one of the strongest depth cues and must be resolved to create a plausible AR scene.

- We can distinguish two cases: a **virtual object** may be either **in front** of or **behind** a **real object**.
- The case where a virtual object is behind a real object, proper handling of occlusions is more difficult. However, *we need a strategy to distinguish visible from occluded virtual objects.*

Occlusion handling: ghost technique

- The basic algorithm to achieve this effect uses **phantoms** (**ghost technique**).

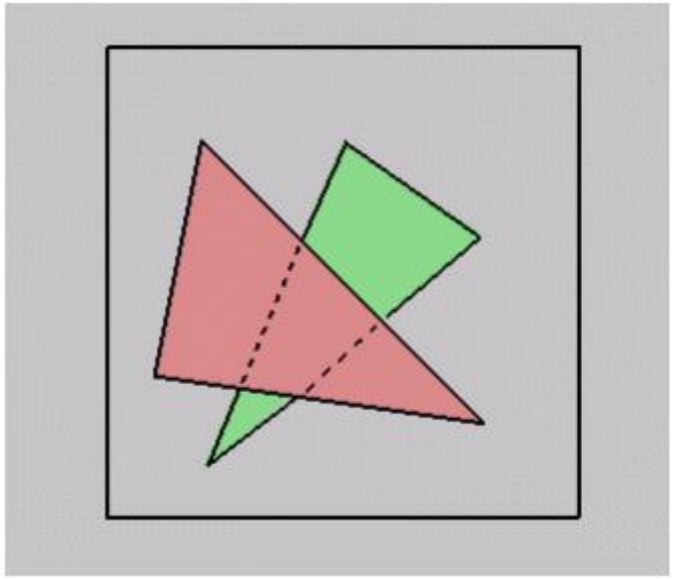
- Phantom rendering makes use of the standard **z-buffer** (depth buffer) capability of a modern graphics processing unit (GPU).
- A phantom is a **virtual representation** of a **real object**, which is rendered **invisibly** (only the **z-buffer** is modified).
- This establishes **correct depth values** for the **real object** visible in the video.
- If the **phantom** objects are correctly **registered**, then hidden fragments from the virtual objects can be rejected by the **standard z-buffer algorithm**.

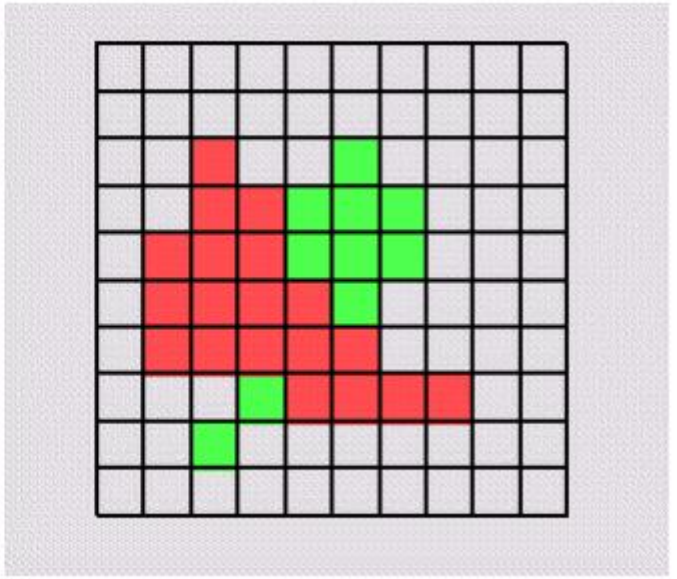
(parenthesis) Z-buffer

We have learnt how to go from 3D to 2D with the visualization pipeline

But a real (3D) scene is composed of different objects.

What happens if multiple primitives occupy the same pixel on the screen? Which is allowed to paint the pixel?





Z-buffer

idea: retain depth (Z in eye coordinates) through projection transform

– use canonical viewing volumes

– each vertex has z coordinate (relative to eye point) intact

Z-buffer algorithm

augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel

– at frame beginning, initialize all pixel depths to ∞

– when rasterizing, interpolate depth (Z) across polygon and store in pixel of Z-buffer

– suppress writing to a pixel if its Z value is more distant than the Z value already stored there

Z-buffer algorithm

```
// First of all, initialize the depth of each pixel.

d(i, j) = infinite // Max length

// Initialize the color value for each pixel to the background color

c(i, j) = background color

// For each polygon, do the following steps :

for (each pixel in polygon's projection){

    // Find depth i.e, z of polygon

    // at (x, y) corresponding to pixel (i, j)

    if (z < d(i, j)){

        d(i, j) = z;

        c(i, j) = color;

    }

}
```

Z-buffer pros and cons

Pros

-easy to implement

-exploit GPU

Cons

-high memory consumption (e.g. 1280x1024x32 bits)

with 16 bits cannot discern millimeter differences in objects at 1 km distance!!!

Occlusion handling: ghost technique

- The basic algorithm to achieve this effect uses **phantoms** (**ghost technique**).

- Phantom rendering makes use of the standard **z-buffer** (depth buffer) capability of a modern graphics processing unit (GPU).
- A phantom is a **virtual representation** of a **real object**, which is rendered **invisibly** (only the **z-buffer** is modified).
- This establishes **correct depth values** for the **real object** visible in the video.
- If the **phantom** objects are correctly **registered**, then hidden fragments from the virtual objects can be rejected by the **standard z-buffer algorithm**.

Occlusion handling: ghost technique

- The following algorithm explains the details:
  1. Draw the video image to the color buffer.
  2. Disable writing to the color buffer.
  3. Render phantoms of the real scene, only to the z-buffer.
  4. Enable writing to the color buffer.
  5. Draw virtual objects.

- *We will see how to handle occlusions and to apply the ghost technique in libraries.*

Occlusion handling: ghost technique

Ghost or phantoms technique has the following prerequisite: having the z-buffer of the real scene, i.e. having a 3D model of the real scene (or at least of the real object) perfectly aligned with virtual representation

Occlusion handling : to acquire depth values for the real object

- Several techniques have been developed to facilitate obtaining a **depth map of the scene dynamically**, either through special hardware measures or by making certain assumptions and processing steps.

- On one end of the spectrum of techniques, **object segmentation** may rely on user input to select a foreground object or on *automatic image processing*. The system tracks the contour of this object in successive frames and uses this information to compute correct occlusions with this object.

Occlusion handling : to acquire depth values for the real object

- On the other end of the spectrum, the most obvious approach to fully automatically obtaining depth maps in real time is the use of a dedicated **depth sensor** (RGB-D cameras).

- The depth information can also be acquired by using a **3D model** of the real object and by performing a computation of its pose (e.g., Vuforia can use *Model Targets* to handle occlusion).

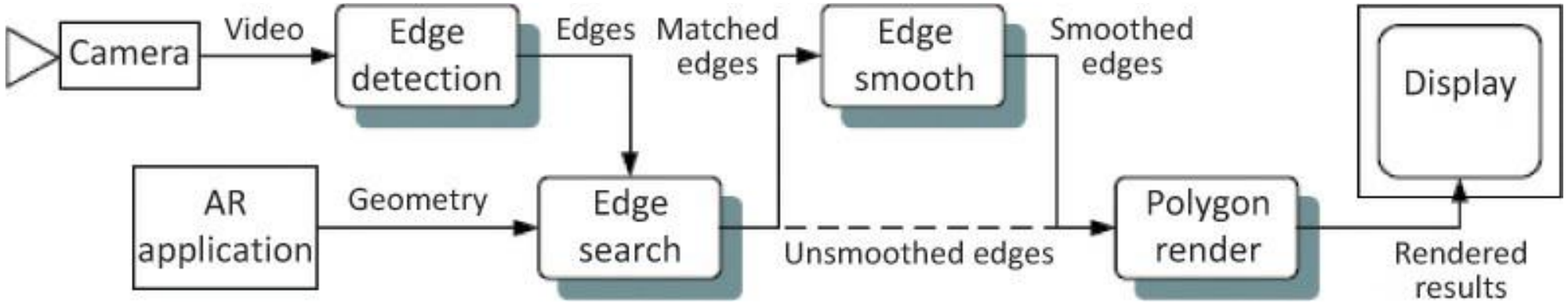
Occlusion refinement

- The **quality** of occlusion achieved using the phantom rendering approach depends on the quality of the input data. The main sources of **inaccuracy** are:
  - Virtual **models** that do not faithfully represent their real-world counterparts cannot produce correct occlusion masks.
  - Incorrect **static registration** between the virtual and real-world coordinate systems means that phantom objects will be rendered at a wrong location or with a wrong orientation.
  - Incorrect **dynamic registration** resulting from tracking errors can deteriorate an otherwise correct static registration, as the camera pose is estimated incorrectly.

- Humans are quite good at detecting such inconsistencies: thus, we need **corrections**, i.e. occlusion refinement.

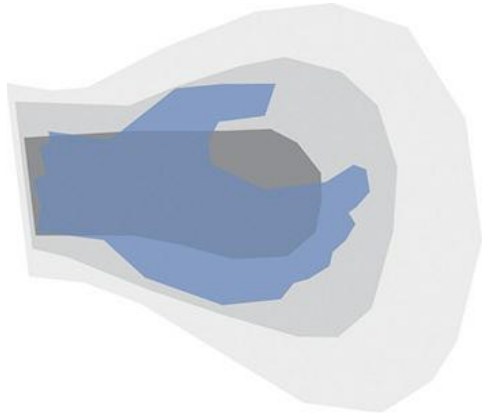
Occlusion refinement

- The main idea of occlusion refinement is that, in general, only the **silhouettes** of a **phantom object** must be accurate to yield correct occlusion between nonpenetrating virtual and real objects.
- Consequently, only the **edges** of a polygonal model that form the silhouette, must be corrected. This correction can be estimated in image space.



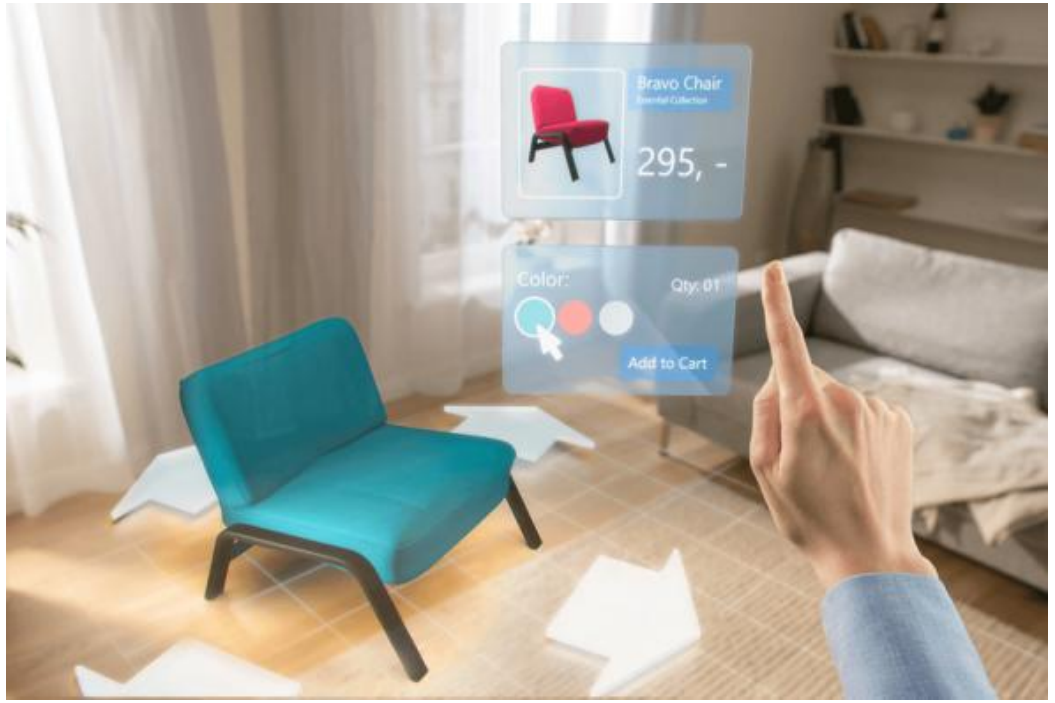
Probabilistic occlusion

- When we deal with **articulated models** such as a moving human, for areas where no precise tracking can be established, such as human hands, a probabilistic model is use.
- This model consists of *multiple nested surfaces*, which become more transparent from innermost to outermost shell.



Ground plane detection

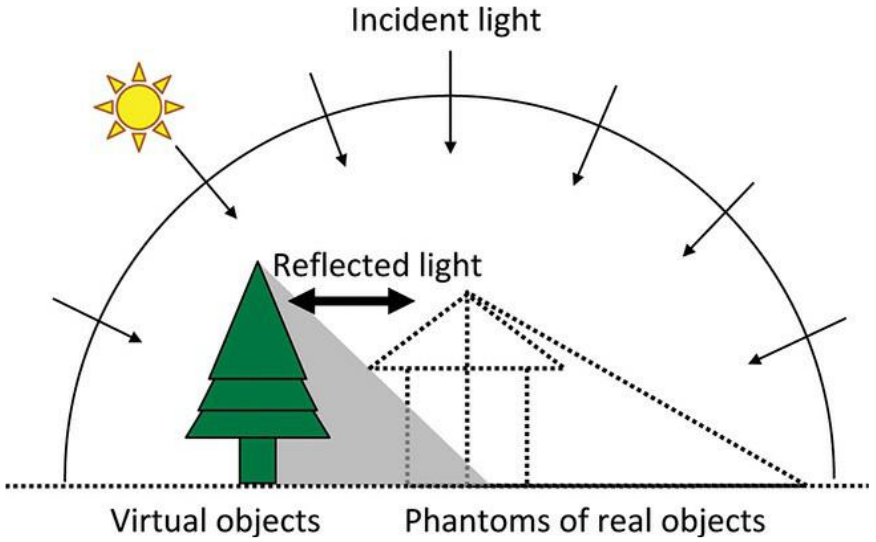
- To provide a visual coherence is also important to recognize the **horizontal** (or vertical) **surfaces** in the environment, such as floors and tabletops.
- Such a detection enables **digital content** to be **placed** in the **correct position** in the environment.
- Both Vuforia and Google ARCore provide this functionality (*we will see them in next Labs*).



Photometric registration

- We also have to compute photometric registration, which lets us resolve how **virtual objects** can be **consistently illuminated** with respect to real objects.

- To achieve this aim, we need to know not only the geometry of virtual and real objects, but also the **incident illumination of the real scene**, that is, the light sources.
- We can consider that for typical small AR workspaces all **light sources** are **distant** and can be treated as external to the scene.



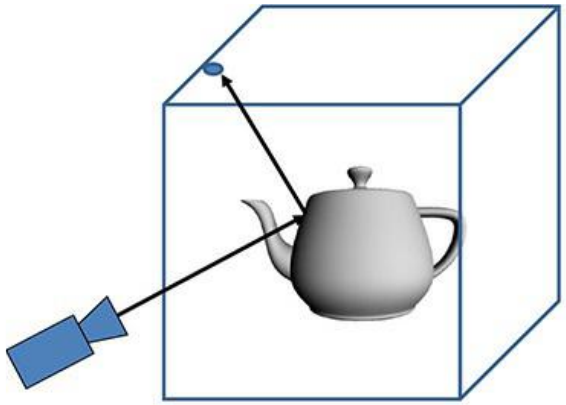
Photometric registration

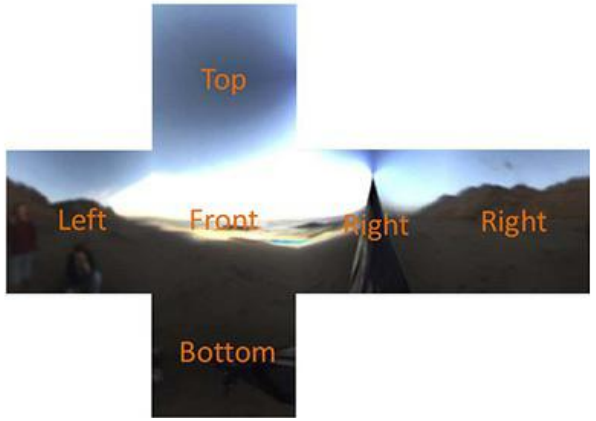
- Considering **local illumination** means that only the light transport from the source to a surface point in the scene is considered.
- In contrast, **global illumination** also considers complex light interactions with other objects in the scene. Therefore, global illumination naturally produces reflection, refraction, and shadowing effects. The computational load of global illumination is high.

- Thus, we consider local illumination in AR.

Photometric registration

- All incoming light can be modeled as **directional light** and stored in a two- dimensional table, i.e. an **environment map**, indexed purely by direction, but not by position in the scene.
- If the environment map represents incoming light (radiance) from the point of view of an observer, it is called a **radiance map**.

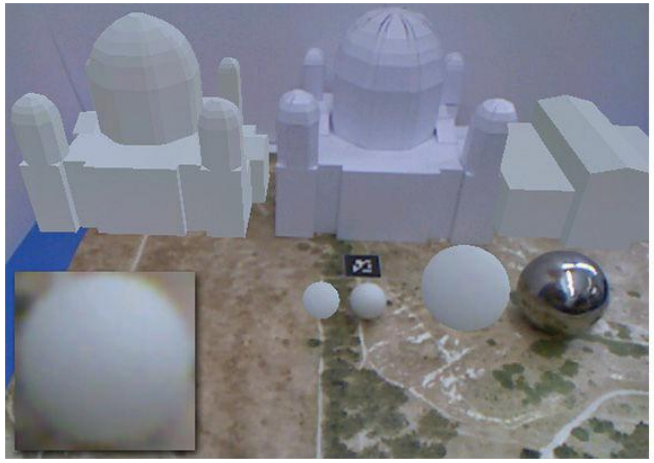


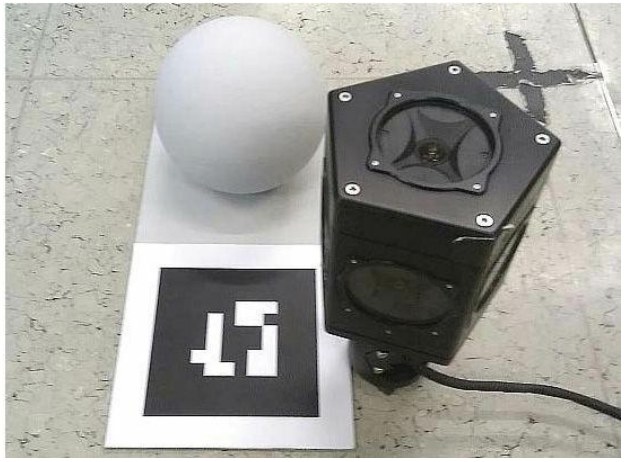


To use environment maps for **image-based lighting**, it is usually necessary to use **high dynamic range** (HDR), which implies representing the illumination in environment maps in meaningful physical units with **floating-point precision**.

Photometric registration: light probes

- The acquisition of a radiance map is effectively achieved with a light probe.
- This tool can take the form of either
  - a **passive light probe** (a reflective “gaze object” placed in the scene and observed by a camera) or
  - an **active light probe** (a camera placed in or near the scene: *both Vuforia and Google ARCore provide this functionality*).





Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

3D Models Representation Methods

Voxel Grid

Voxel grid representation is a method for modeling 3D objects where the space is divided into a regular grid of cubes, known as voxels (volume elements).

Each voxel in the grid can store information such as color, density, or material properties, allowing for a detailed volumetric representation of the object.

This makes voxel grids particularly useful in applications where volumetric data is naturally produced, such as medical imaging (e.g., MRI and CT scans), where organs and tissues are represented as a 3D array of voxels.

Voxel grids are pivotal in computer graphics and game development, exemplified by their use in destructible environments and procedural generation in games like 'Minecraft'. These applications leverage the simplicity and flexibility of voxel grids to create and manipulate 3D environments dynamically.

Point Cloud

A point cloud is essentially a set of data points in space, which represents the external surface of an object or a scene.

These points are typically defined by their coordinates in a 3D space (x, y, z) and can sometimes include additional information such as color, intensity, and normal vectors.

Point clouds are generated by 3D scanning technologies like LiDAR (Light Detection and Ranging), photogrammetry, and depth cameras, which capture the shape of objects and environments with high precision.

Point clouds are simple but they are often sparse, unstructured, and can contain noise, making processing tasks like segmentation, recognition, and reconstruction more complex.

Mesh

It is composed of vertices, edges, and faces that form a polyhedral shape.

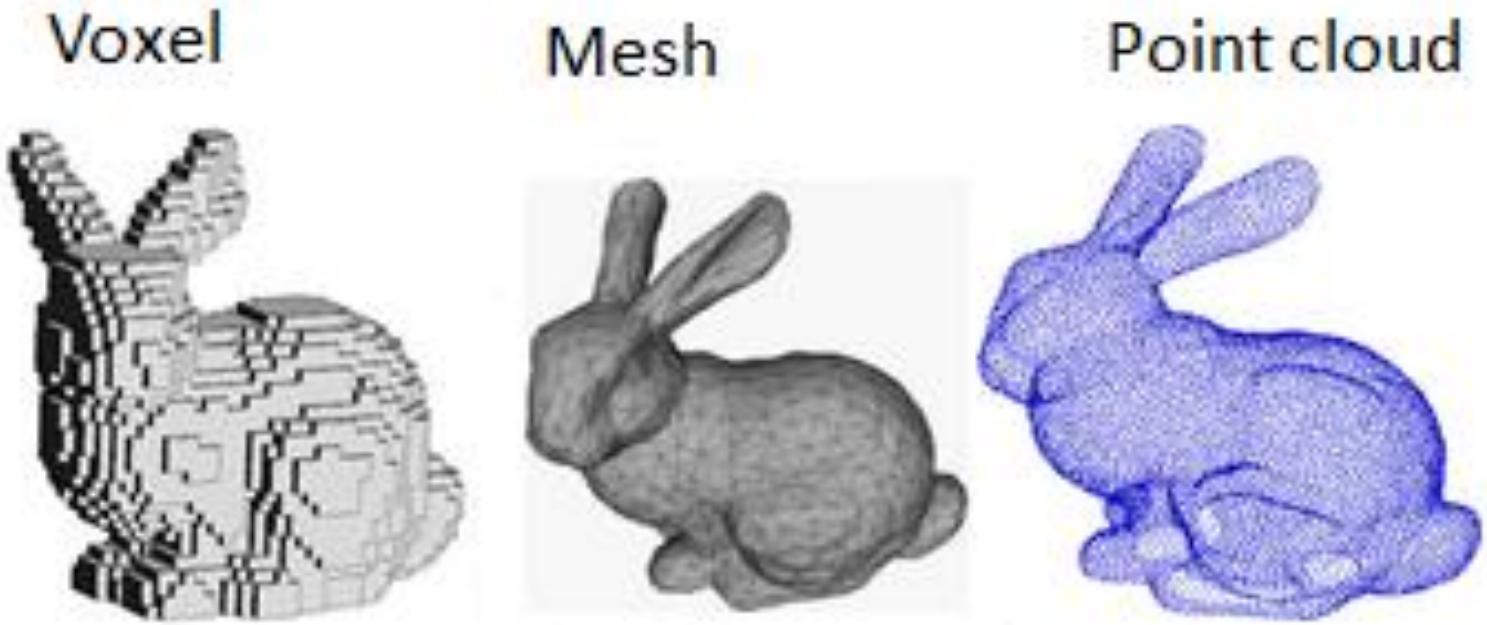
The vertices are point clouds in 3D space, the edges connect pairs of vertices, and the faces (typically triangles or quadrilaterals) are defined by edges connecting three or more vertices.

They can approximate complex geometries with arbitrary precision by adjusting the number of vertices and faces.

Meshes are well-supported by graphics hardware, making them efficient for rendering and simulation tasks.

The flexibility of mesh representation allows for efficient computation of surface properties, such as normals and curvature, which are crucial for realistic rendering and physical simulations.

Voxel, mesh and point cloud



Models in Unity

https://docs.unity3d.com/Manual/working-with-gameobjects.html

https://docs.unity3d.com/Manual/models.html