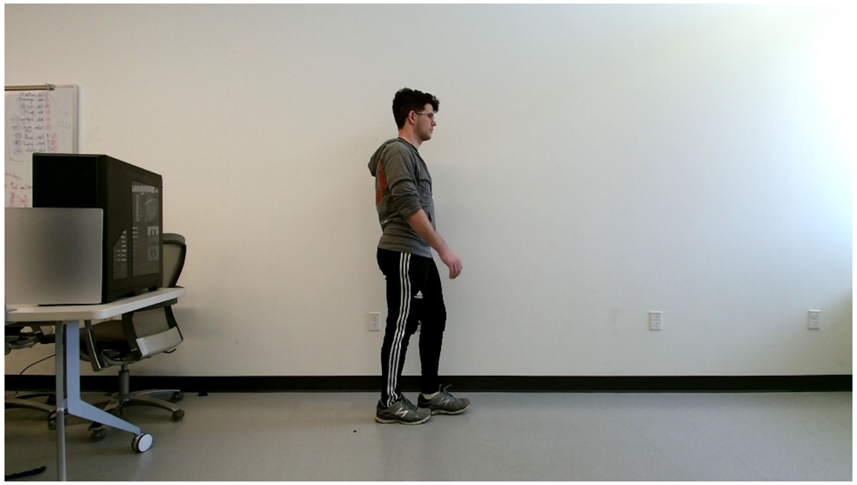
CSE 6367: Computer Vision

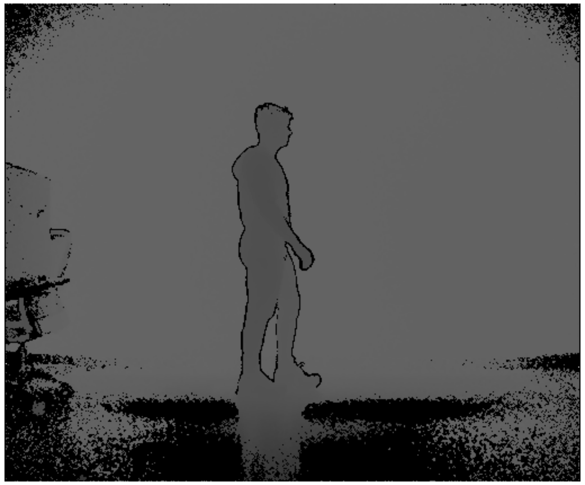
Depth Cameras

Slide Courtesy: Vassilis Athitsos
CSE@UTA

Example of Color + Depth

- Since the first Kinect cameras that came out in 2010, color + depth cameras have become popular.
  - Color + depth cameras are also called RGBD sensors, or RGBD cameras.
- Typically, an RGBD sensor is a camera that simultaneously produces two images:
  - a color image.
  - a depth image.

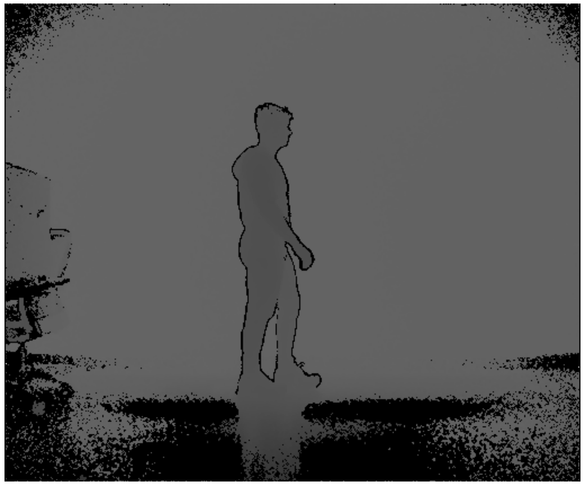

color image


depth image

Depth Images

- The depth image shows, for each pixel, the depth of that pixel.
- The depth is the z coordinate of the 3D point that is visible at that pixel, assuming that the camera pinhole is at depth z=0.
  - The z axis is is perpendicular to the image plane of the depth camera.
- The depth value ranges from 0 to some max value.
  - This max value depends on the sensor.
  - Depth values are usually 16-bit integers.

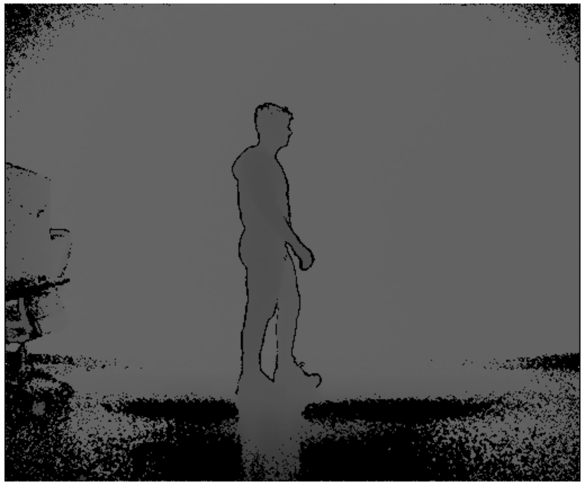
color image


depth image

Dealing with Depth Images

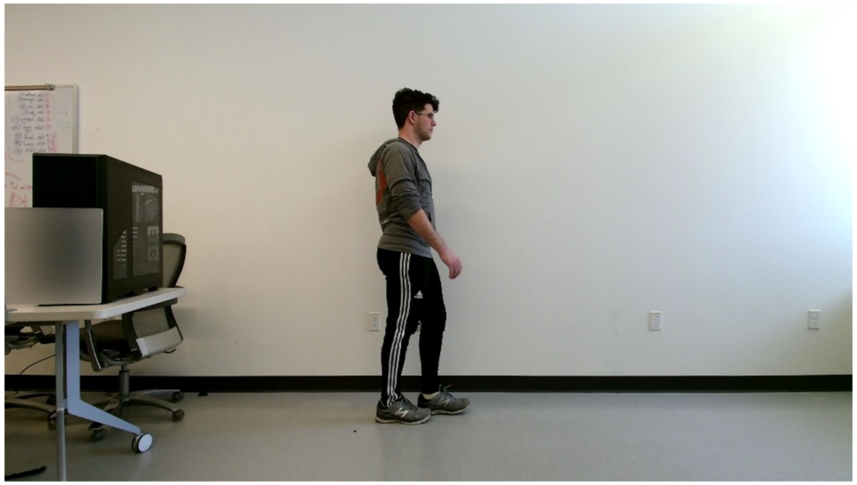
- Since depth values are usually 16-bit integers, existing software and libraries oftentimes do not know how to deal with them.
- To circumvent this problem, sometimes people convert depth images to 8-bit images.
  - Conversion can be done by dividing all values by a constant.
  - Such conversions can lead to problems. Converting from 16 bits to 8 bits (obviously) loses information, that may be useful.

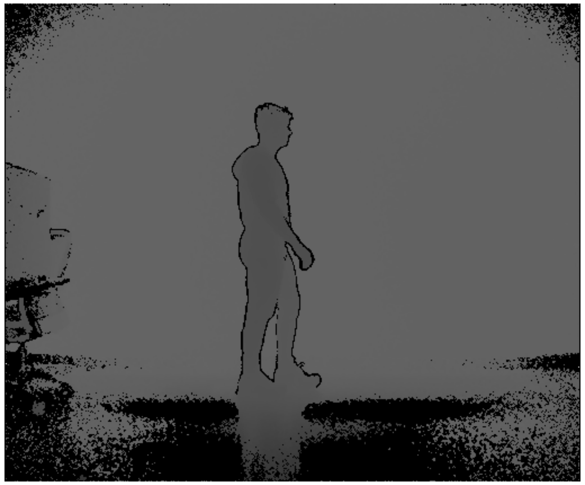
color image


depth image

Pixel Locations Do Not Match

- The two example images were produced by a Kinect v2 camera.
- The two images have different sizes and different aspect ratios.
  - The color image has size 1080x1920.
  - The depth image has size 424x512.
- Thus, finding the location in the color image that corresponds to a specific location in the depth image is not straightforward.
  - We will make an attempt to do that later in these slides.

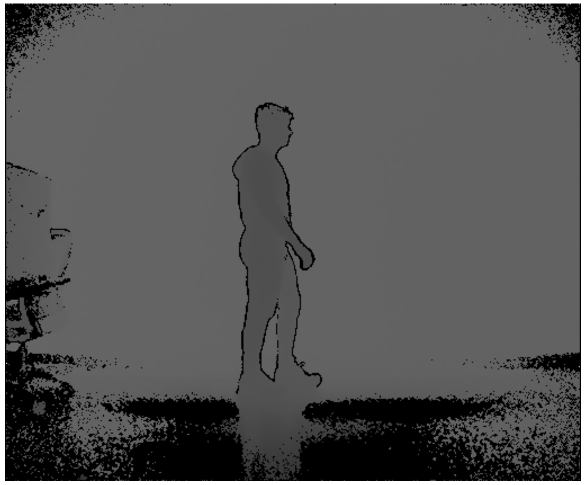

color image


depth image

Visualizing a Depth Image

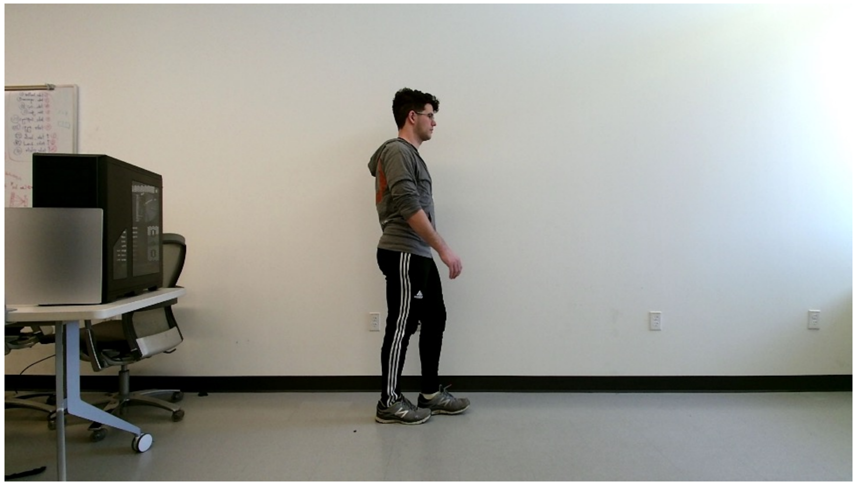
- The TIF image format supports 16-bit images.
  - Probably some other formats also provide similar support.
- Matlab's imread and imwrite functions work correctly with such images, representing them as 16-bit 2D array.
- Displaying the depth image, we see that darker values correspond to smaller depth.

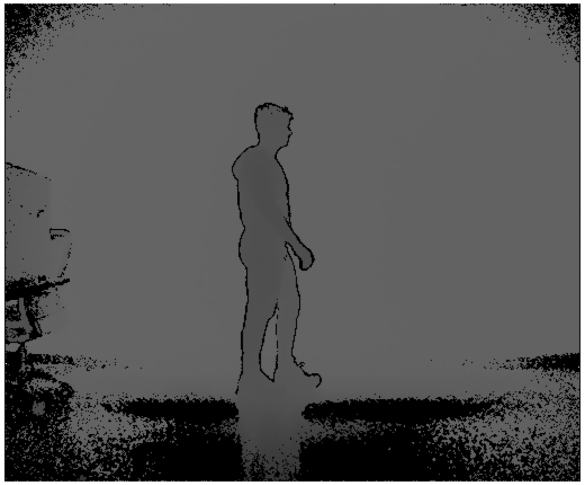
color image


depth image

Visualizing a Depth Image

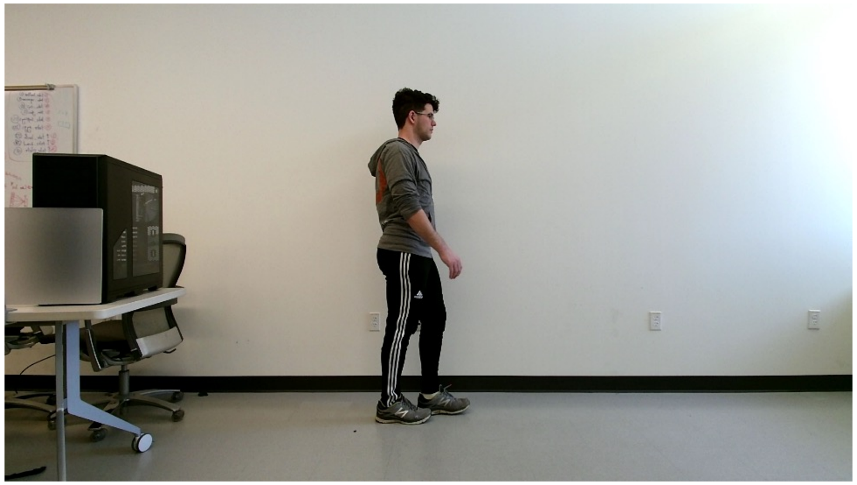
- Depth sensors can have a special value to indicate that the depth of a pixel was not estimated successfully.
  - For Kinect v2, that value is 0.
  - So, depth 0 simply means that the depth is invalid.
  - We see examples of depth 0 on the floor, and also around the person.
- We should be mindful of these values, and not treat them as regular depth values.

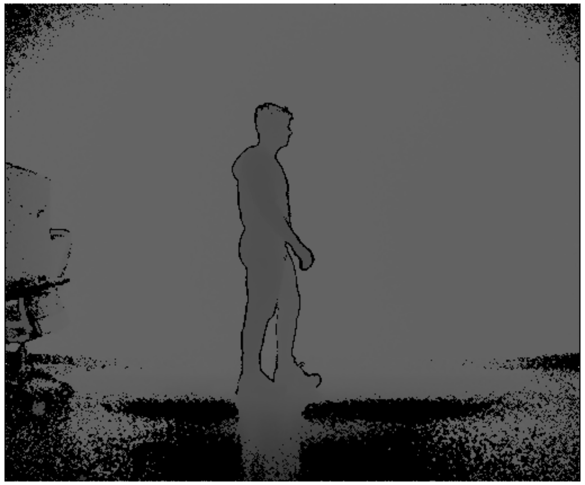

color image


depth image

Visualizing a Depth Image

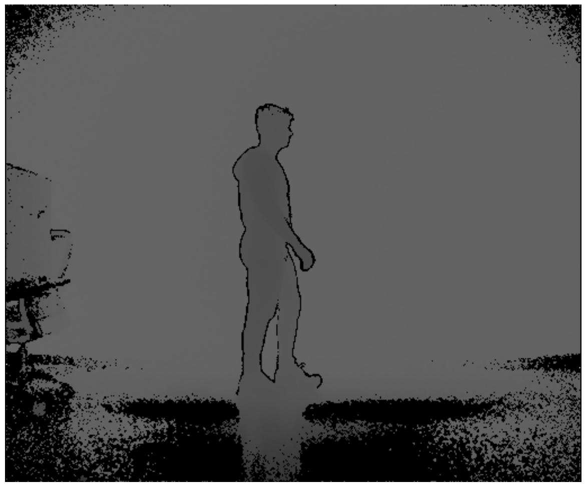
- Displaying a depth image the standard way can make it difficult to visualize the richness of the information that is contained.
- In the example depth image:
  - Clearly the person has smaller depth than the background wall.
  - However, depth differences within the person region are hard to see.
- There are various ways to help us visualize better the information contained in a depth image.

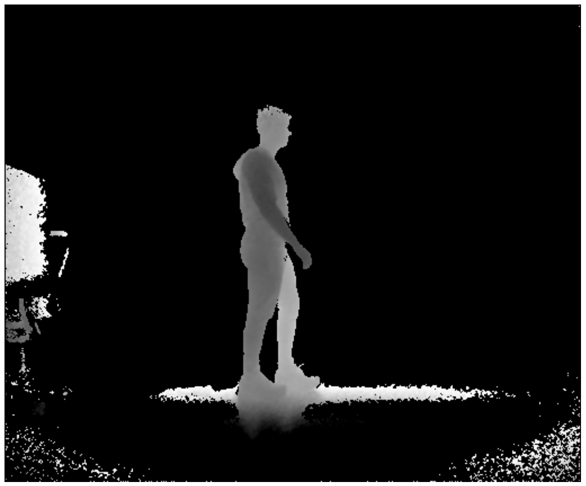

color image


depth image

Visualizing a Specific Depth Range

- We can see more details if we only visualize a specific range of depth values.
- For example: in our depth image below, the depth values in the person region are mostly between 2100 and 2850.
- We can write code to select and amplify that range.
  - Map value 2100 to black, and value 2850 to white.


depth image

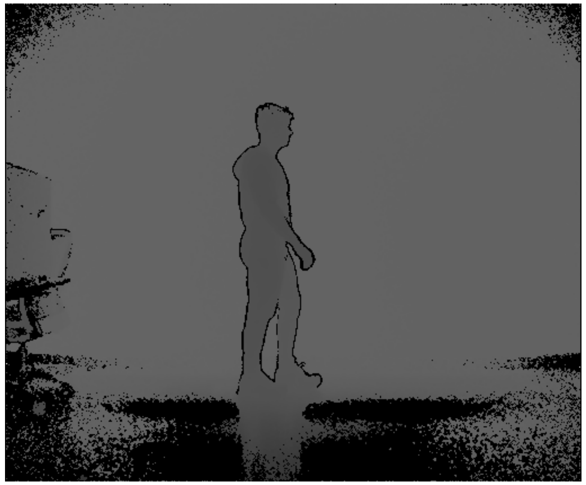

highlighting range 2100-2850

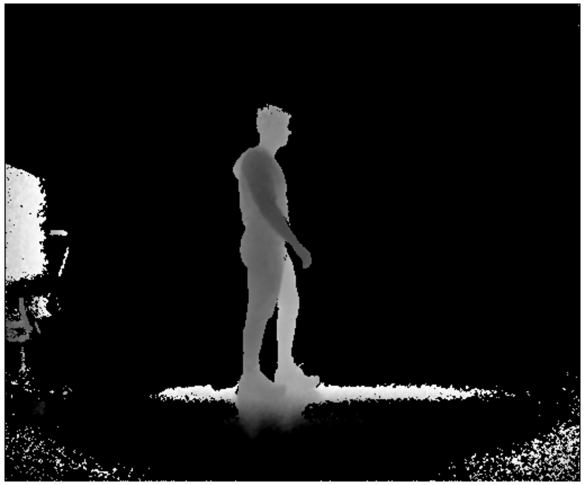
Visualizing a Specific Depth Range

```
function result = highlight_range(image, low, high)

invalid = (image == 0);
result = image-low;
result(image < low) = 0;
result(image > high) = 0;
result(invalid) = 0;
imshow(result, []);
```

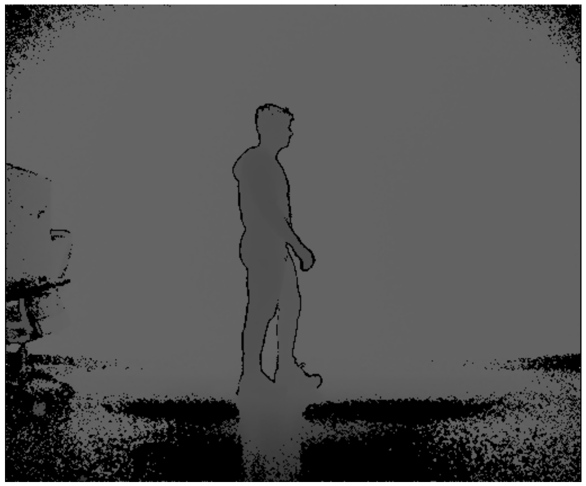
- Pixels whose values are outside the range of interest are set to 0.
- Range [low, high] is mapped to range [0, high-low].

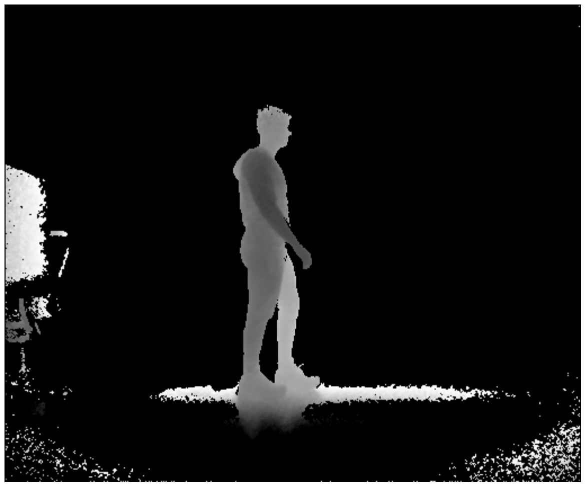

depth image


highlighting range 2100-2850

Visualizing a Specific Depth Range

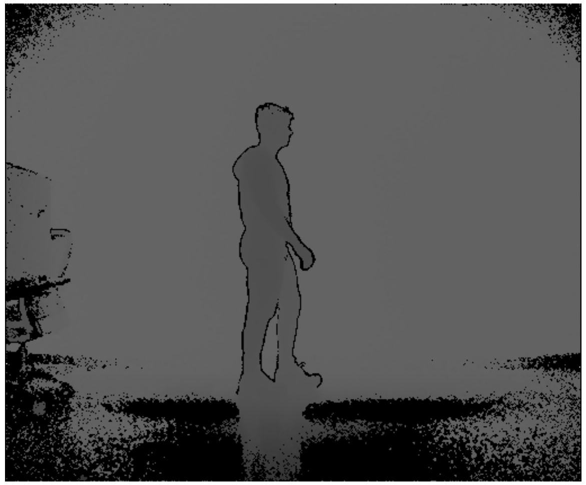
- The resulting image, highlighting a specific depth range, helps us observe depth differences within the person region, that were not easily observable in the original depth image.
  - We see differences in depth among the right arm, the right leg, the head, and the left leg. Those differences were hard to see in the original image.
- This comes at the cost of not seeing any information about pixels whose values are outside that range.

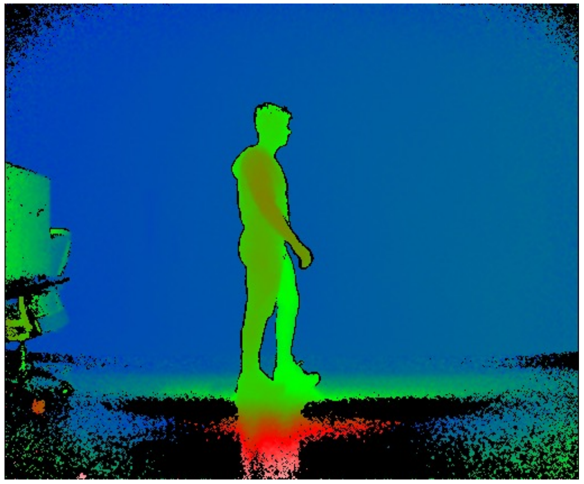

depth image


highlighting range 2100-2850

Mapping Depth to Color

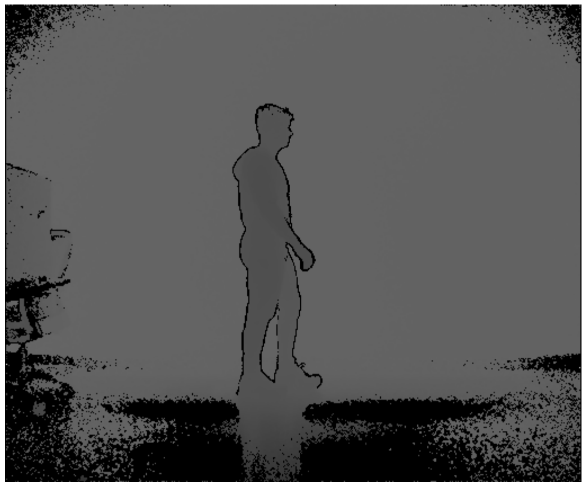
- Another approach for visualizing depth images is to map different depth ranges to different colors.
- Here we see an example where the color sequence is white->red->green->blue. (see all_code/12_depth/color_code3.m)
  - Lowest depth values get mapped to white.
  - Highest depth values get mapped to blue.
- It is easier to note small depth differences in the colorized image.

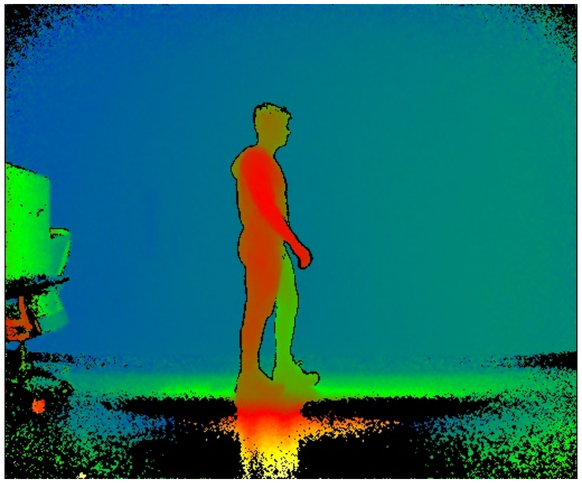

depth image


colorization of depth image

Mapping Depth to Color

- Here is another example, using more colors.
- The color sequence is white->yellow->red->green->blue.
  - This is implemented at all_code/12_depth/color_code.m
- Remember: the only use of mapping depth to color is to help us visualize differences in depth values.
  - More colors make it easier to visualize small differences in depth values.

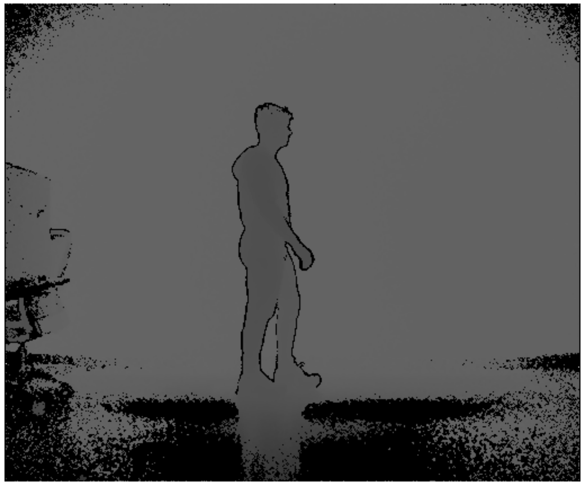

depth image


colorization of depth image

Example of Using Depth Information

- In the depth image shown below, the person can be found by simply choosing an appropriate range of depth values.

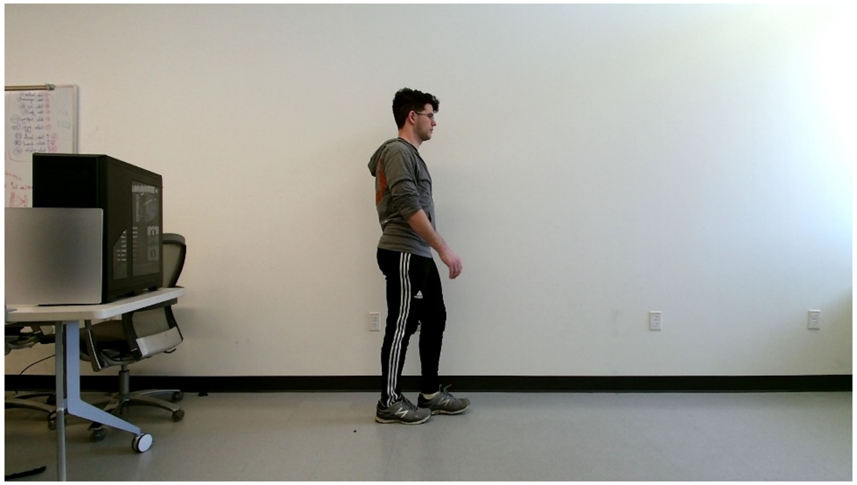
color image

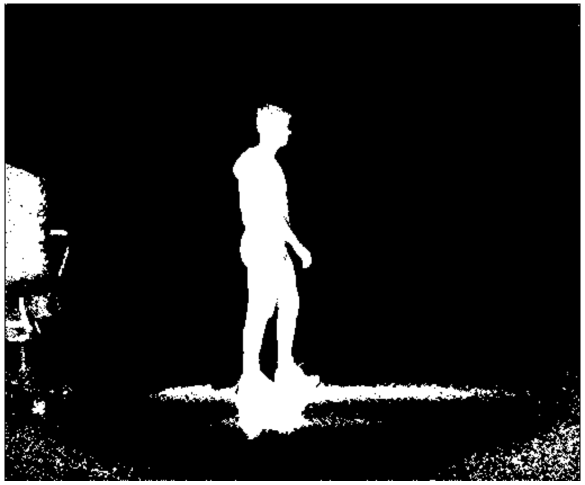

depth image

Example of Using Depth Information

- In the depth image shown below, the person can be found by simply choosing an appropriate range of depth values.

```
mask = ((d > 2100) & (d < 2850));
```

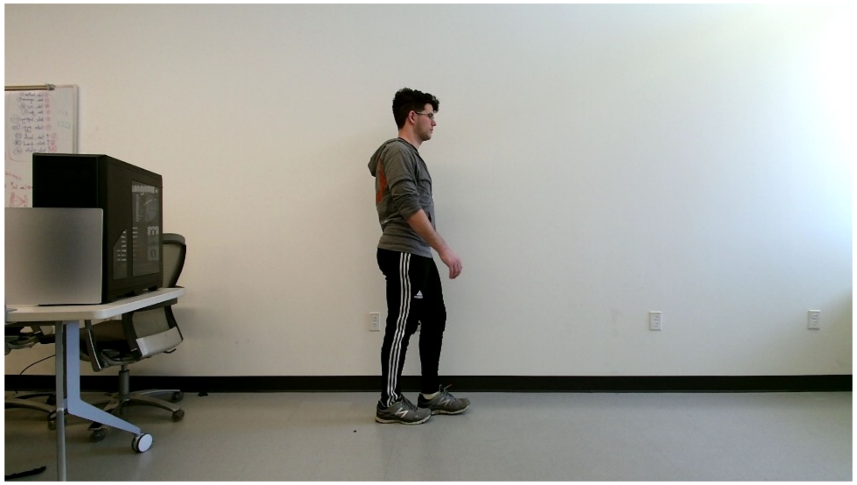

color image

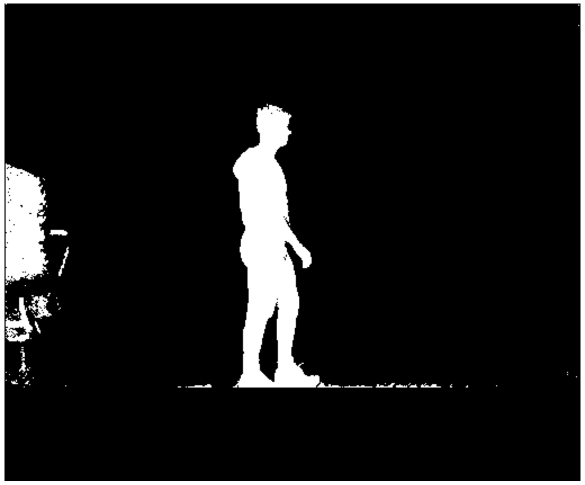

mask

Example of Using Depth Information

- This range also includes some of the floor, but we can clean that up.
- Many libraries for processing depth images can automatically detect the floor. Here, we will remove it manually.

```
mask = ((d > 2100) & (d < 2850));
mask(342:424, :) = 0;
```

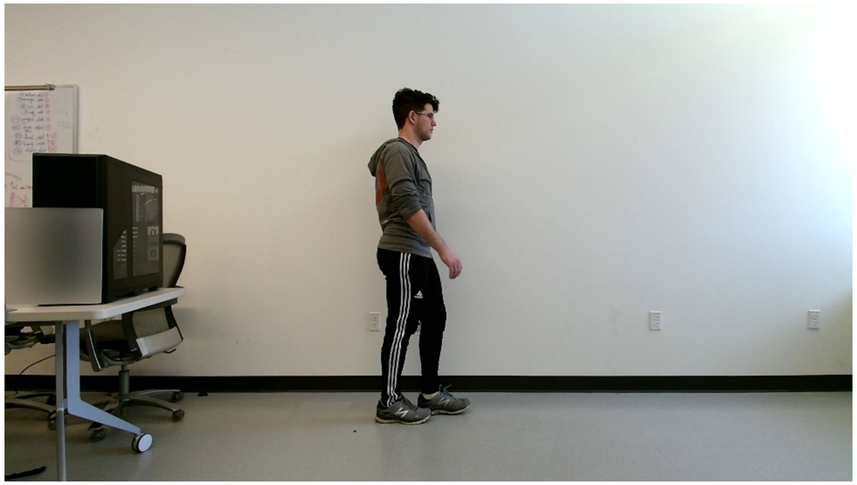

color image

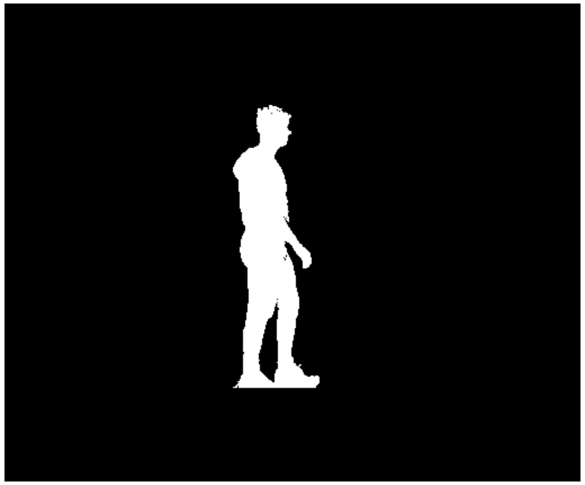

mask after removing floor

Example of Using Depth Information

- Since some other objects have the same depth range, we can identify the person region as the largest connected component.

```
mask = ((d > 2100) & (d < 2850));
mask(342:424, :) = 0;
person = largest_connected(mask);
```


color image

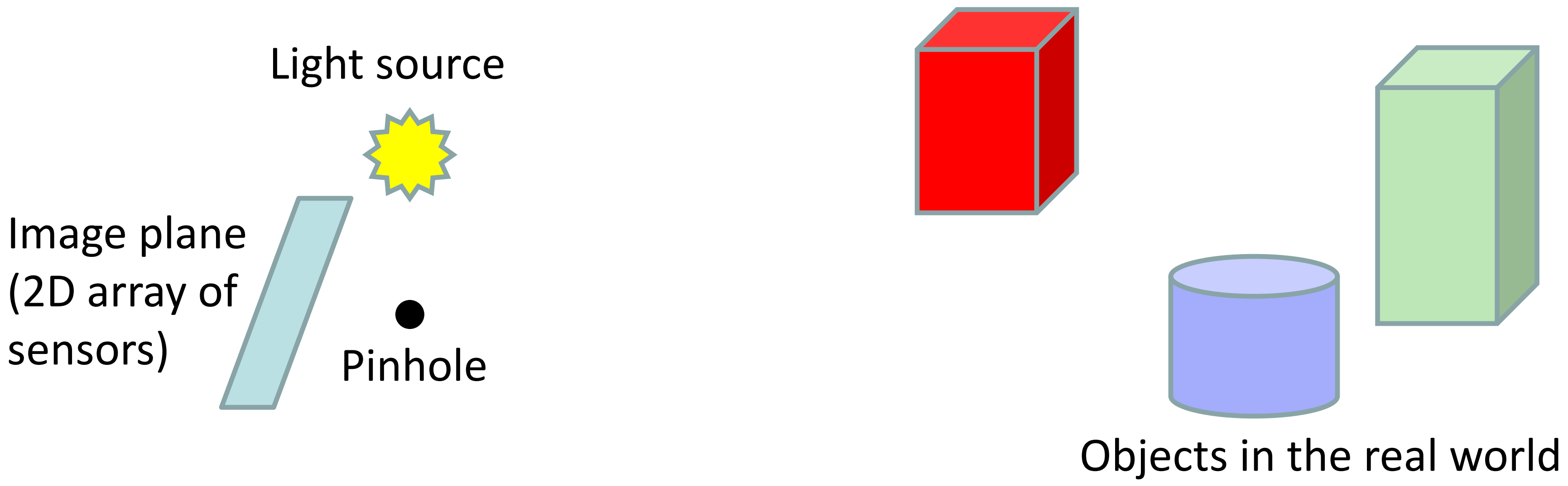

mask after removing floor

How Depth Sensors Work

- In theory, stereo cameras can be used to estimate depth.
- In practice, the accuracy of stereo cameras for depth estimation is inferior to that of alternative technologies, so the most popular depth sensors do not use stereo cameras (at least not in their standard format).
- Various technologies can be used to estimate depth.
- Two popular ones are:
  - Time-of-flight cameras (example: Kinect v2).
  - Structured lighting (example: Kinect v1).

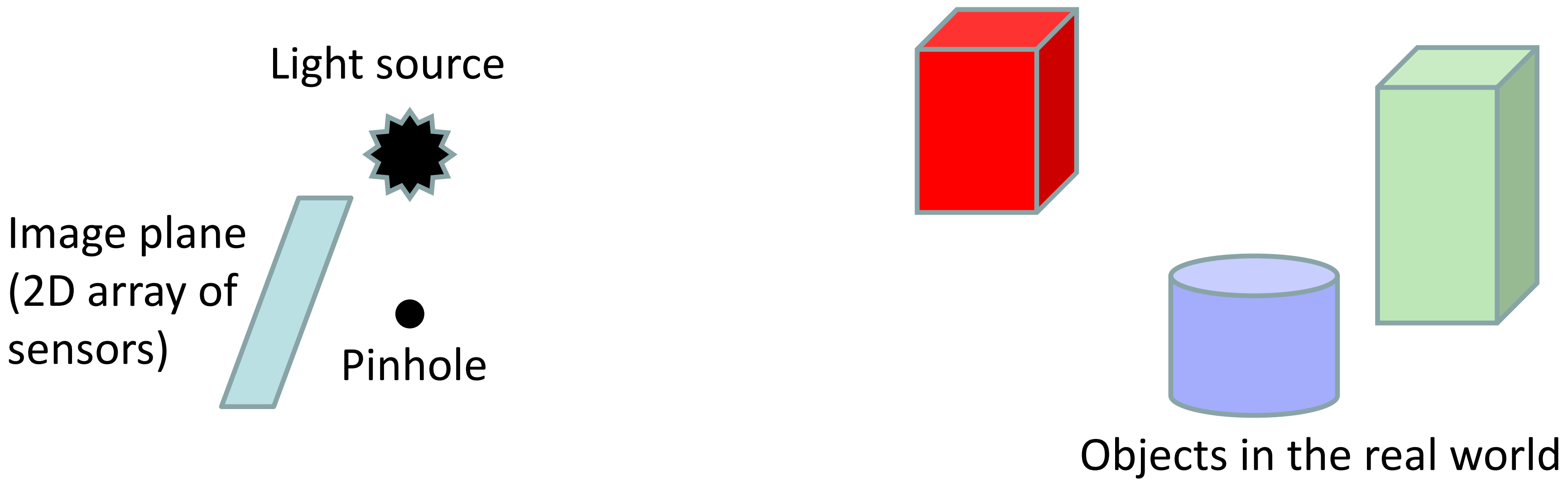
Time-of-Flight (ToF) Cameras

- This is a very simple version of how time-of-flight (Tof) cameras work, ignoring various practical complications.
- Basic parts:
  - A light source that the system can turn on and off as needed.
  - A camera (in the simple case, a pinhole and a 2D array of sensors).



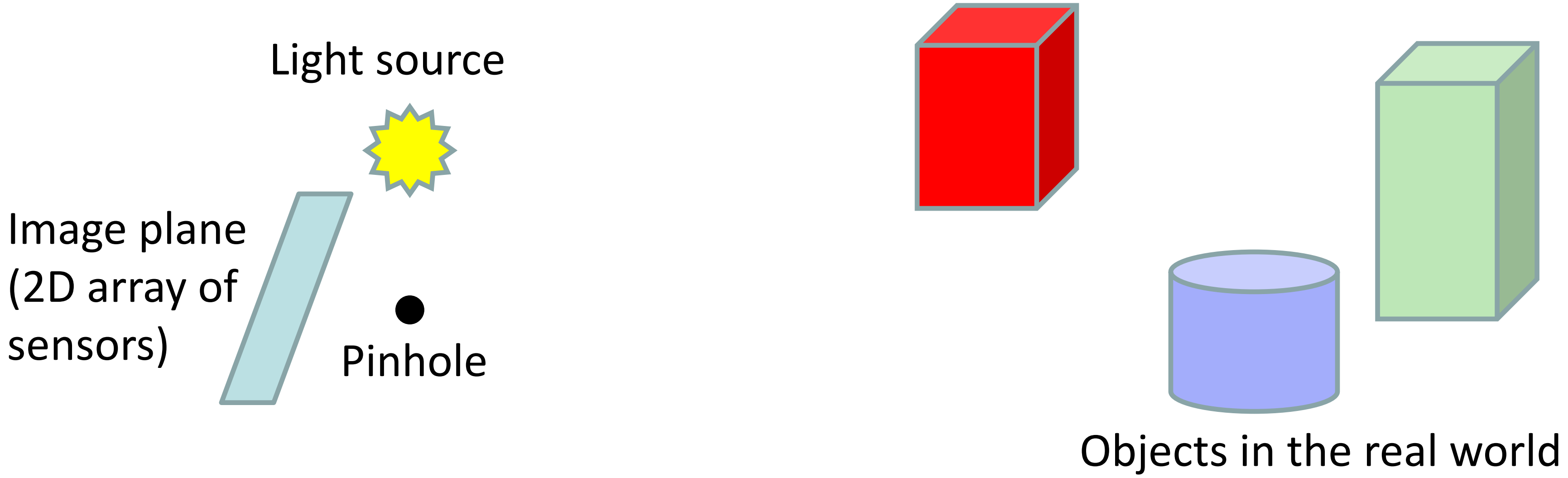
Time-of-Flight (ToF) Cameras

- First, it is dark (the light source is off).
- Consequently, the sensors record low light intensity.



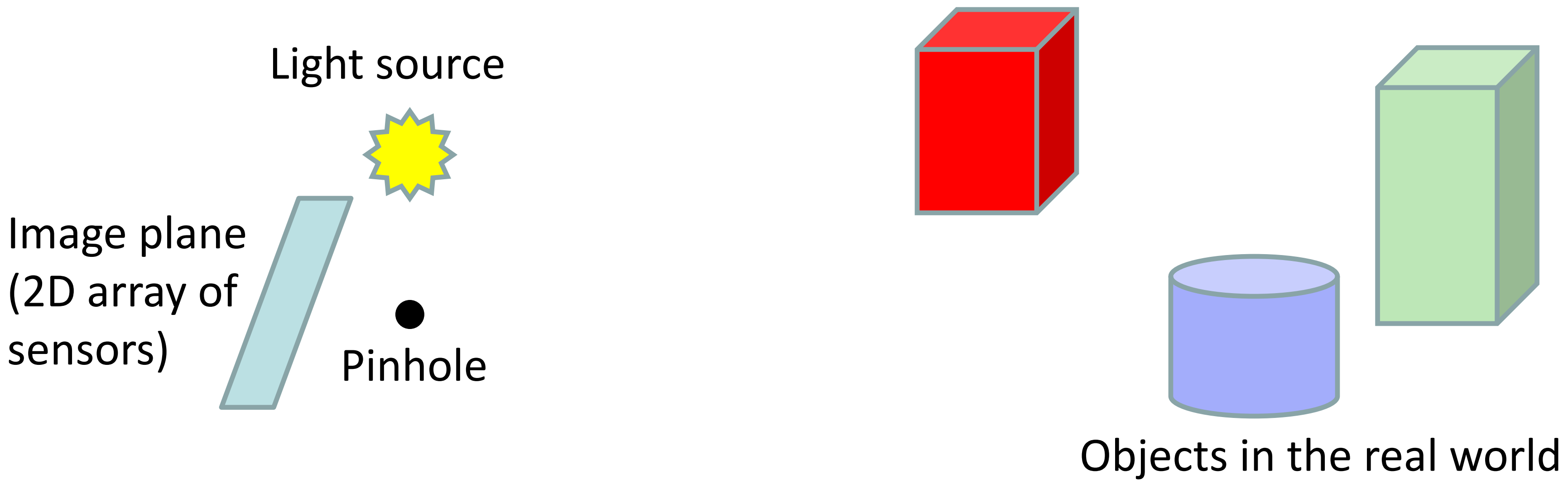
Time-of-Flight (ToF) Cameras

- Then, the light source turns on.
- The light sensors start recording higher light intensity, but **at different times**.
- Why different times?



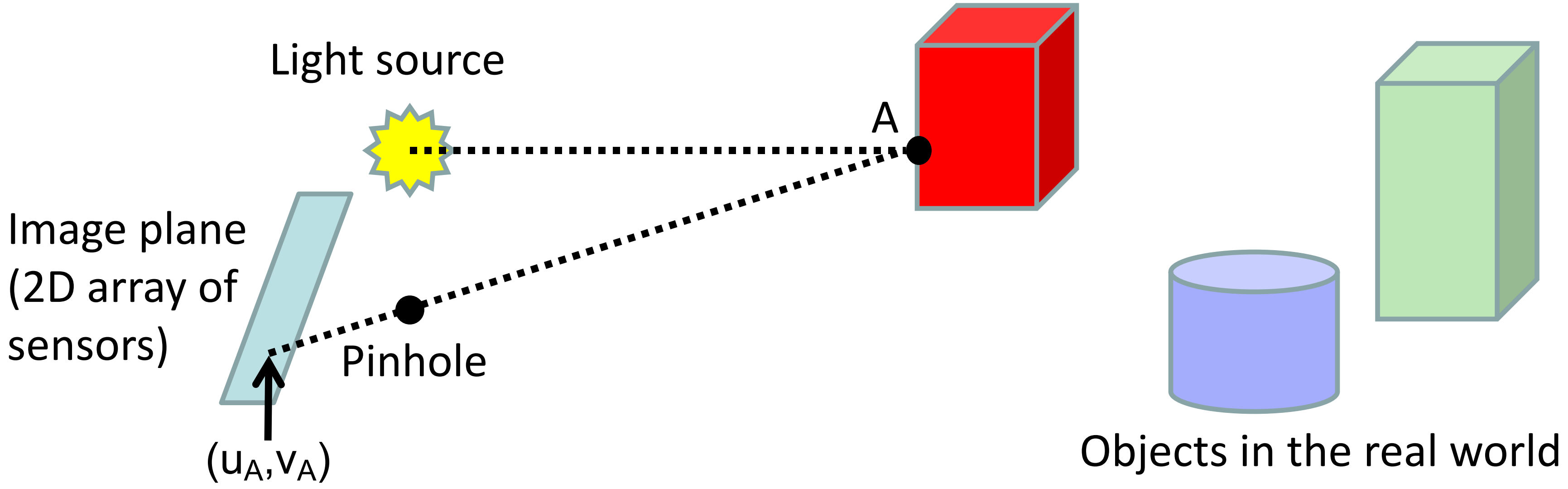
Time-of-Flight (ToF) Cameras

- Then, the light source turns on.
- The light sensors start recording higher light intensity, but **at different times**.
- Why different times?
- Because it takes a different amount of time for the light to reach each sensor.



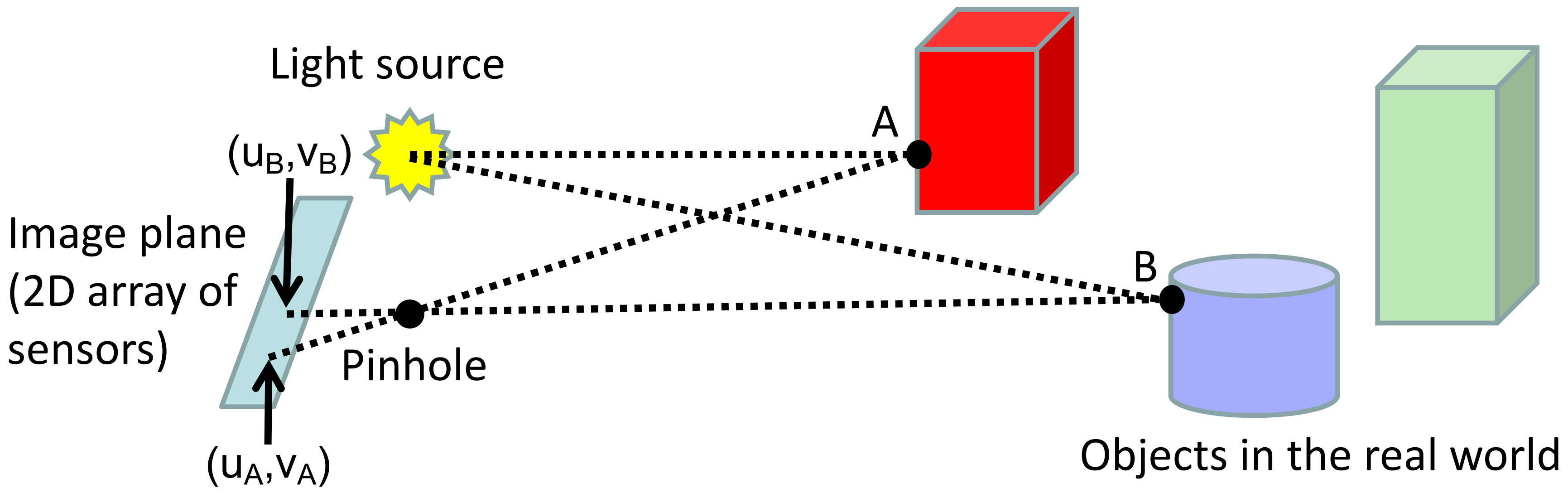
Light Travel from Source to Sensor

- Consider 3D point A.
- From the moment the light source turns on:
  - The light has to travel from the light source to A.
  - The light is reflected from A, and goes through the pinhole to pixel (u_A,v_A).
  - The time that the sensor will start recording light is proportional to the distance from the light source to A plus the distance from A to the sensor for pixel (u_A,v_A).



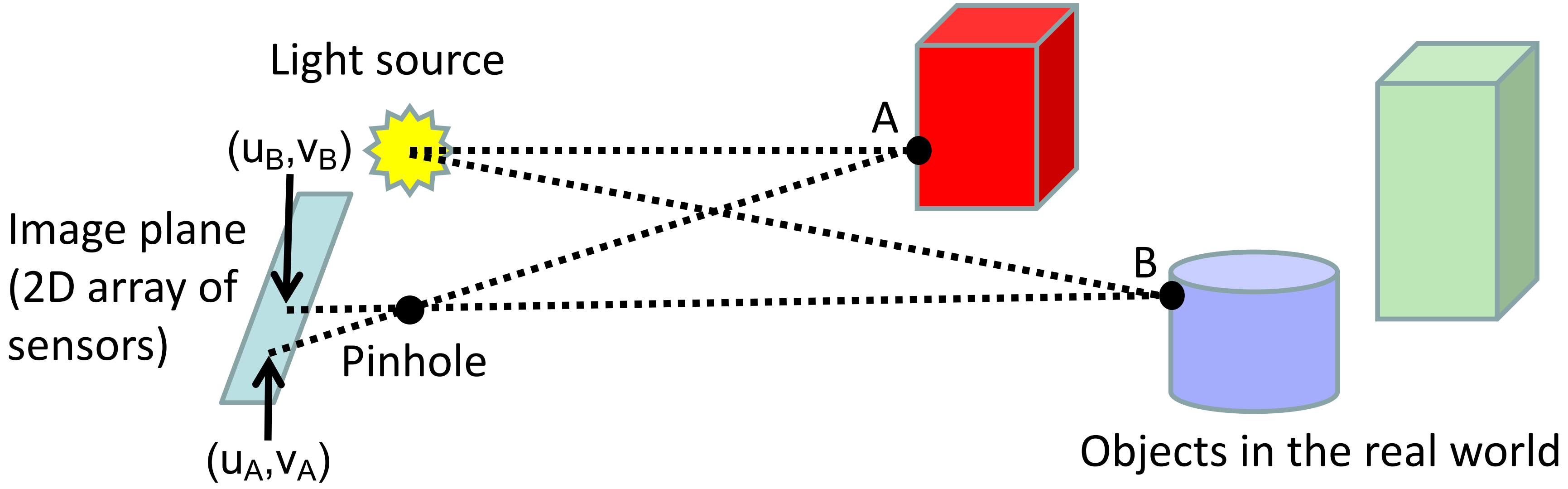
Light Travel from Source to Sensor

- Consider 3D point B.
- From the moment the light source turns on:
  - The light has to travel from the light source to B.
  - The light is reflected from B, and goes through the pinhole to pixel (u_B,v_B).
  - The light will arrive earlier at (u_A,v_A), and later at (u_B,v_B).



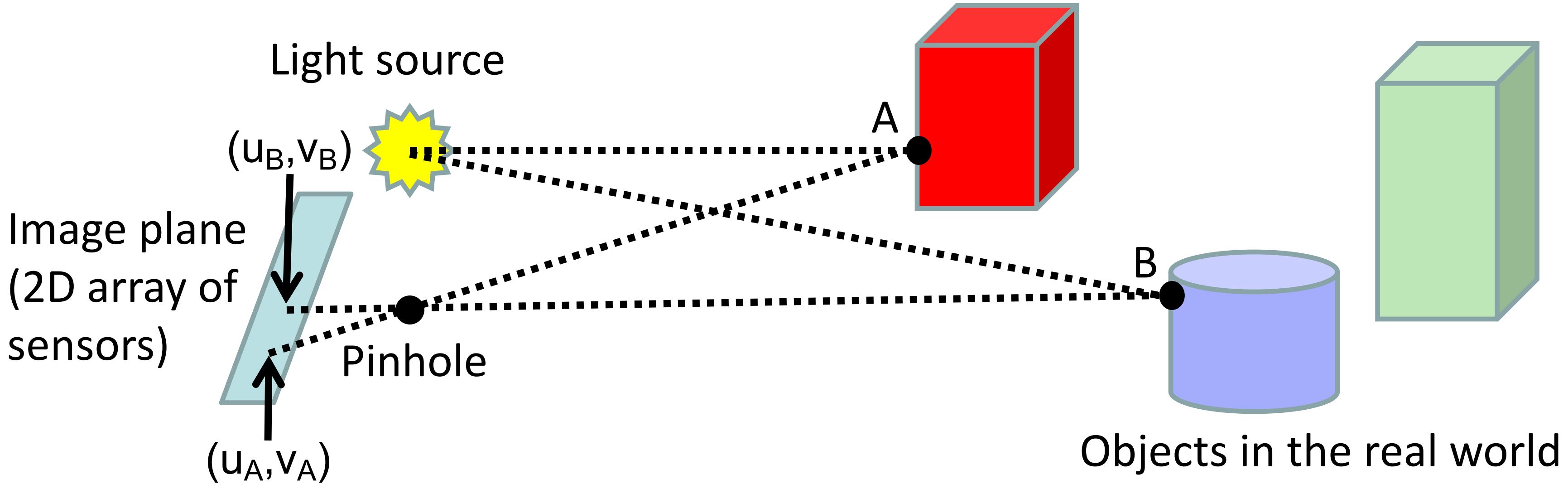
Light Travel from Source to Sensor

- So, the time it takes for the light to reach each sensor (u,v) can be used to measure the distance to the 3D location corresponding to (u,v).



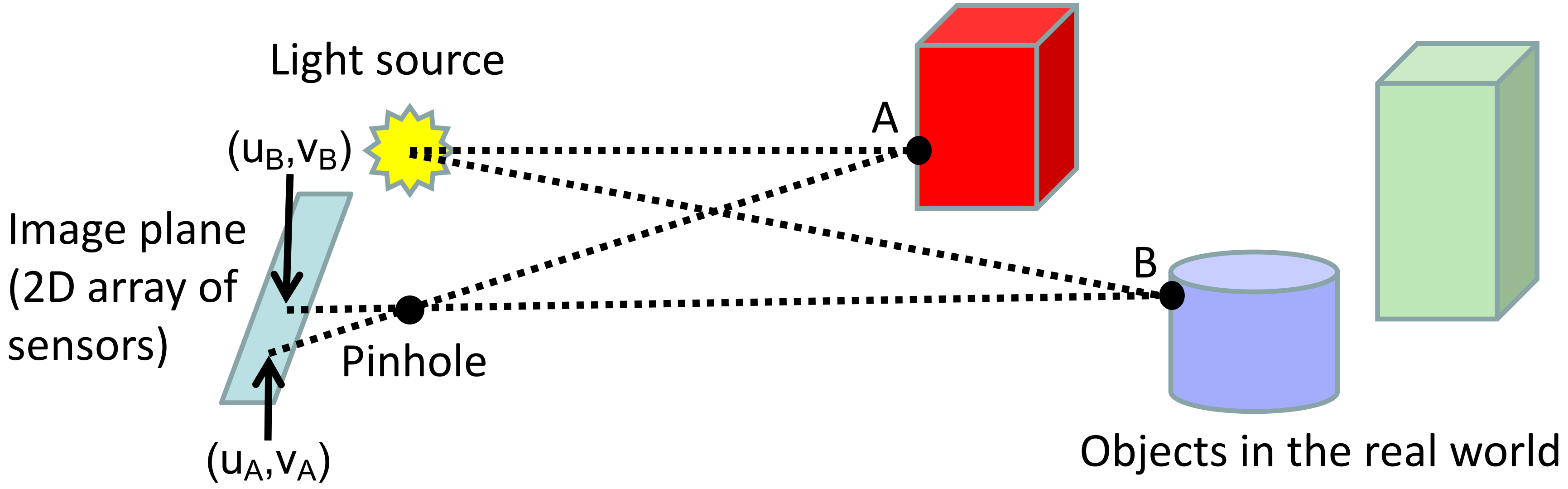
ToF Cameras: Some Complications

- This was a very simplistic description of the principle behind ToF cameras.
- Complication 1: in practice, sensors do not record an exact time when light starts hitting.
  - However, sensors do record the amount of light sensed within a time interval, and that is sufficient for distance calculations.



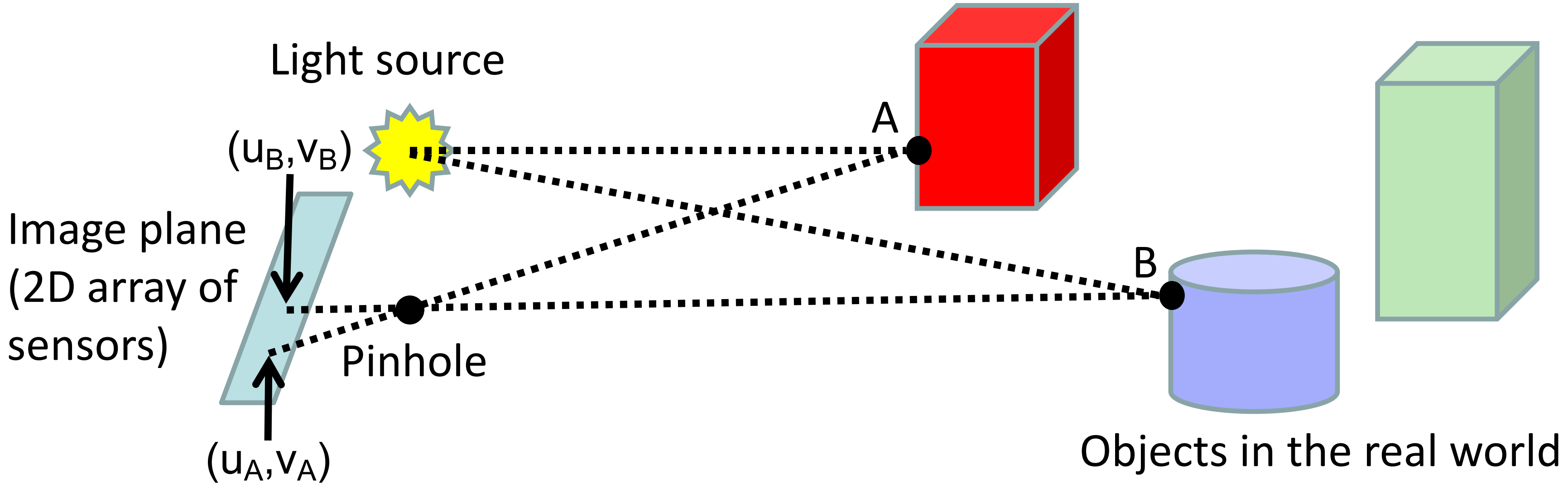
ToF Cameras: Some Complications

- Complication 2: the time it takes for the light to travel is the sum of two distances (light source to 3D point and 3D point to sensor).
- What we really care about is distance from pinhole to 3D point.
- This can be calculated by calibrating the camera, and by computing its location with respect to the light source.



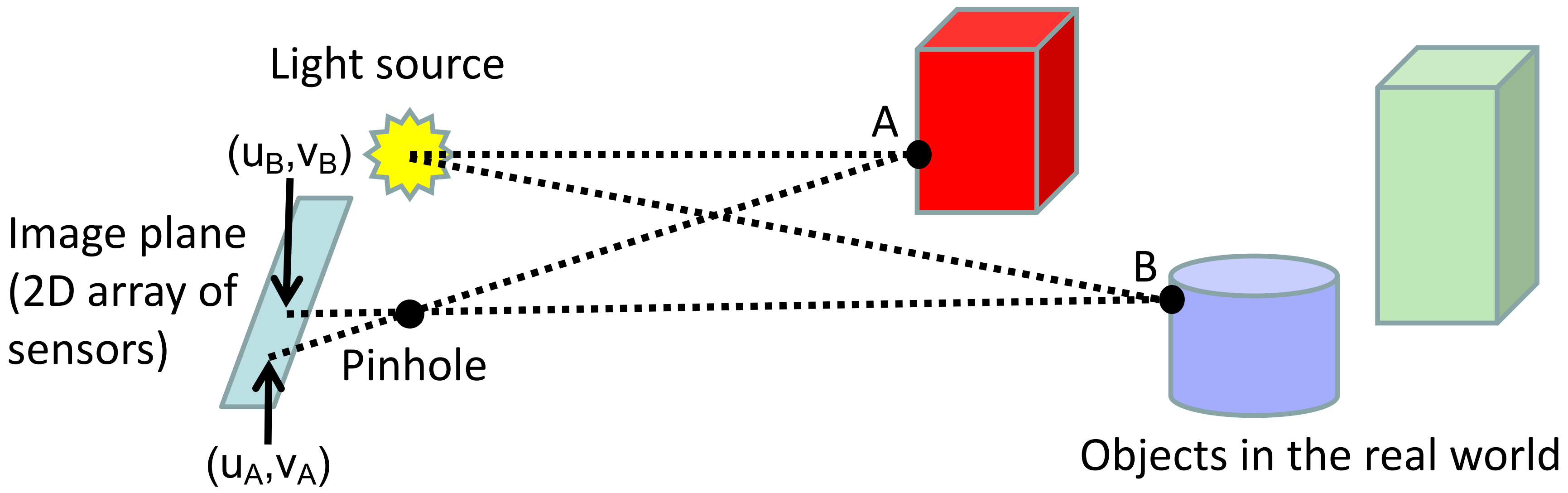
ToF Cameras: Some Complications

- Complication 3: typically the scene is not dark.
  - When the light source is off, other sources of light are still present (sunlight, indoor lighting, …).
  - However, when the light source is on, objects become brighter.
  - The sensor output can be used to estimate when the brightness increases for each point.



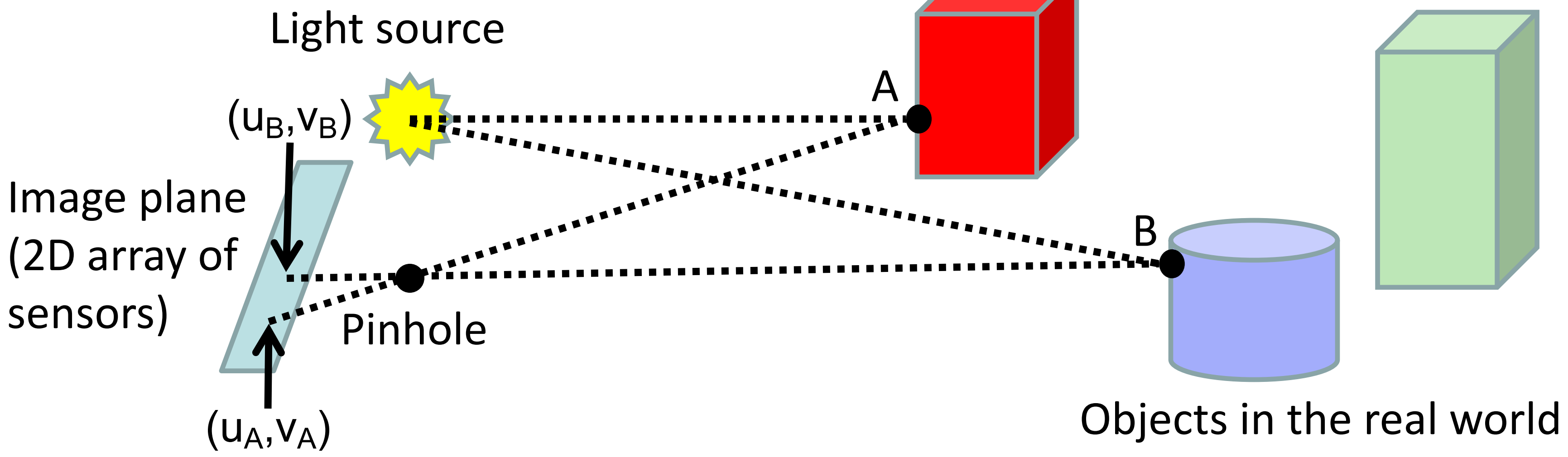
ToF Cameras: Speed of Light

- An interesting note: light travels really fast (300,000 km/sec).
- If the ToF camera is 3 meters (10 feet) away from point A, how long does it take for the light to travel to A and back?
  - 20 nanoseconds.
  - If the distance is 3.15 meters, the time becomes 21 nanoseconds.
- ToF cameras can capture such tiny time differences!



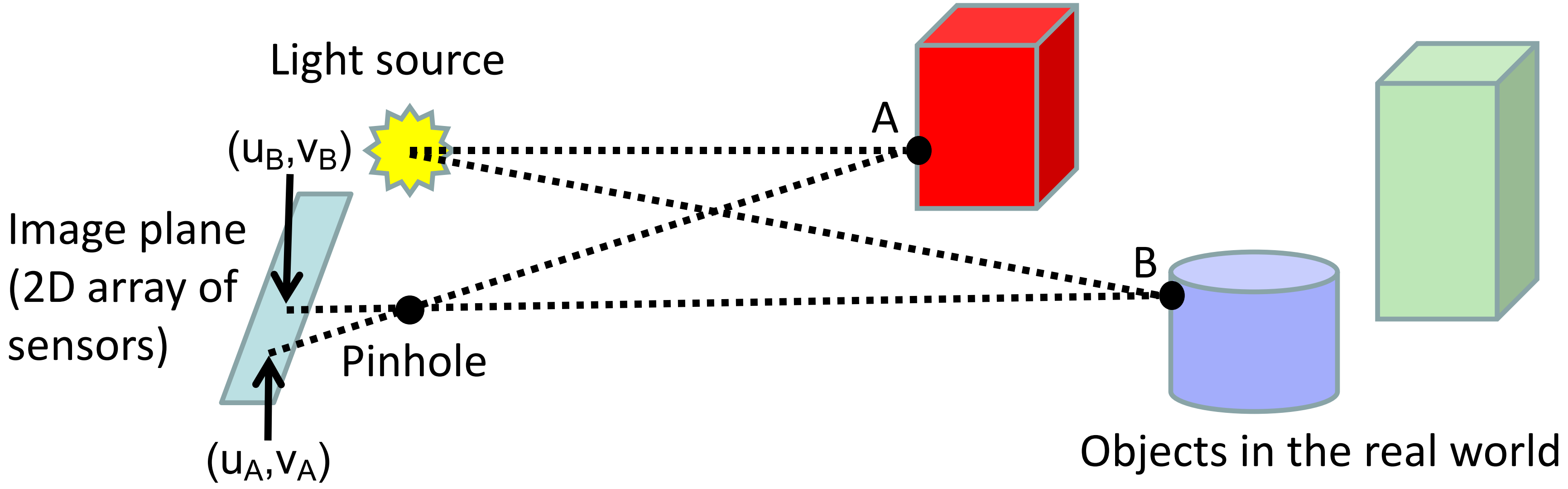
ToF Cameras: Infrared Light

- Projecting light to the scene can be annoying for humans present at that scene.
- Solution: project infrared light.
  - Invisible to humans.
  - We need to use infrared sensors for the camera.



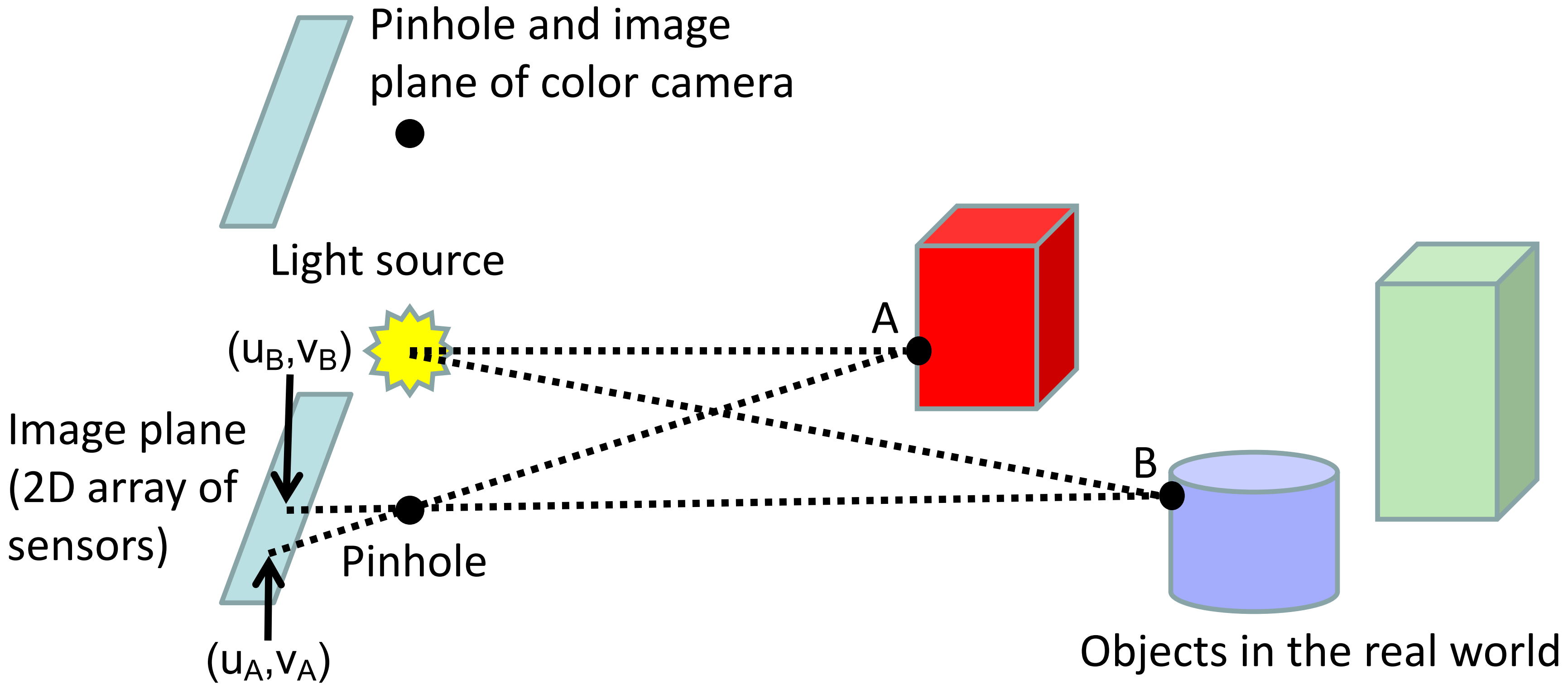
ToF Cameras: Getting Color Data

- The setup we have described so far provides a depth image, no color information.



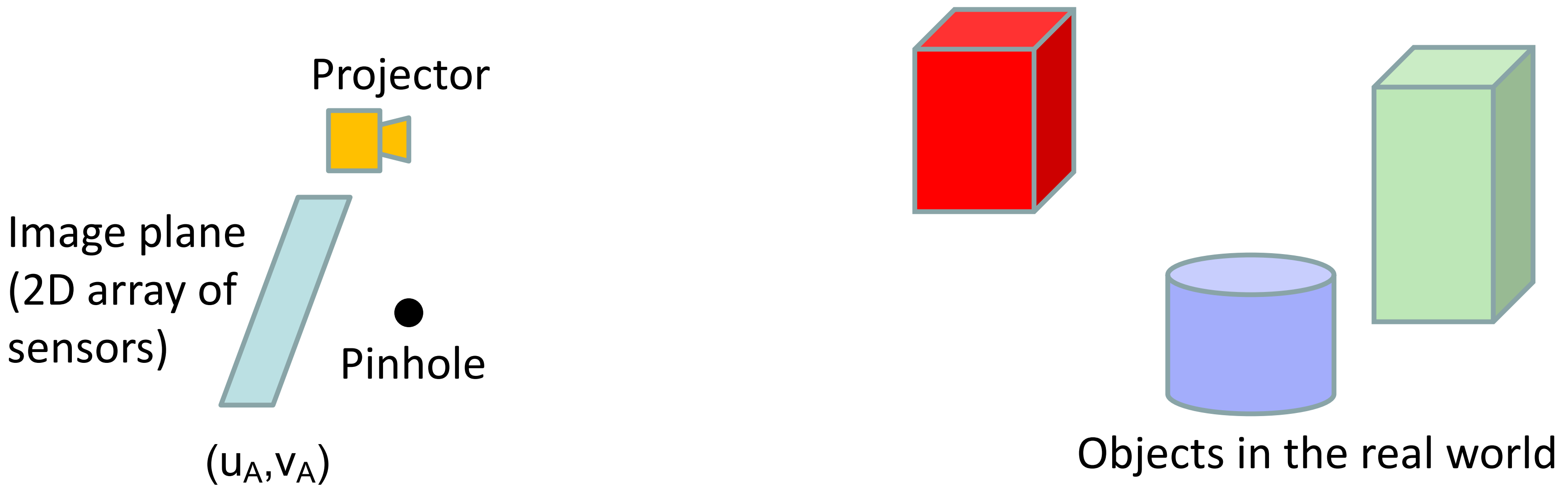
ToF Cameras: Getting Color Data

- To get a color image in addition to the depth image, we simply add a standard color camera to the setup.


Pinhole and image plane of color camera
Light source
A
B
(u_B, v_B)
Image plane (2D array of sensors)
Pinhole
(u_A, v_A)
Objects in the real world

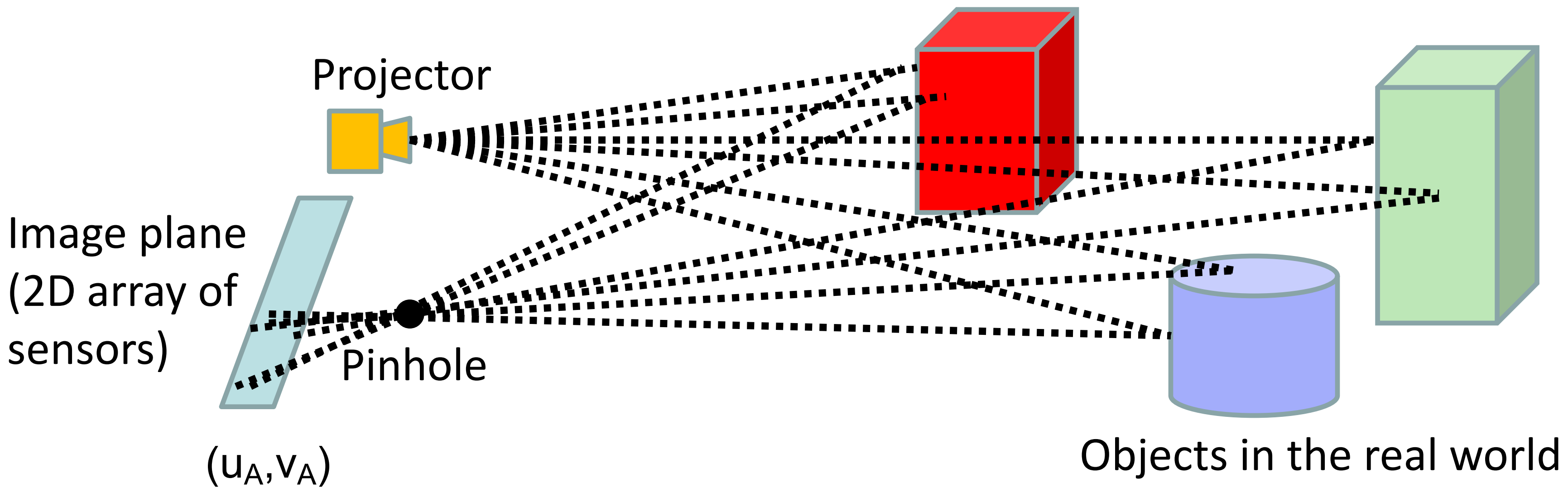
Structured Light

- Structured light is an alternative technology for estimating depth.
- Key components:
  - A projector, projecting a pattern into the scene.
  - A camera (in the simple case, a pinhole and a 2D array of sensors).



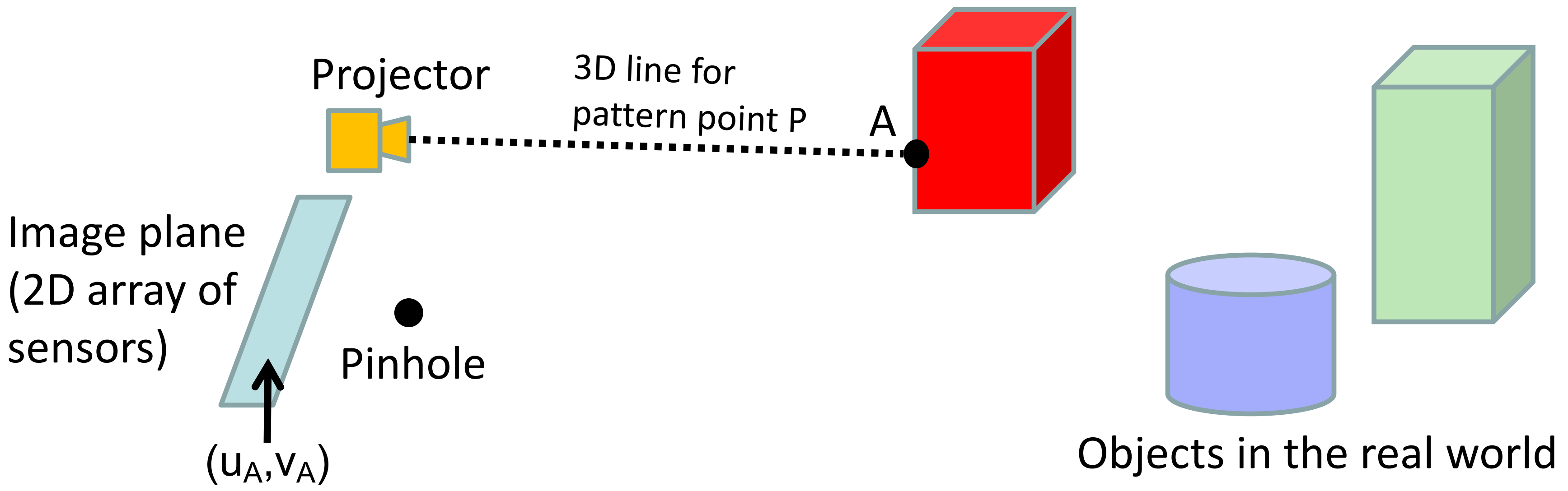
Structured Lighting

- The projector projects a specific pattern into the scene.
- The camera captures an image of the scene.
- Software in the camera detects the pixel coordinates of the various points of the pattern.
- Stereo estimation estimates the depth of each point.



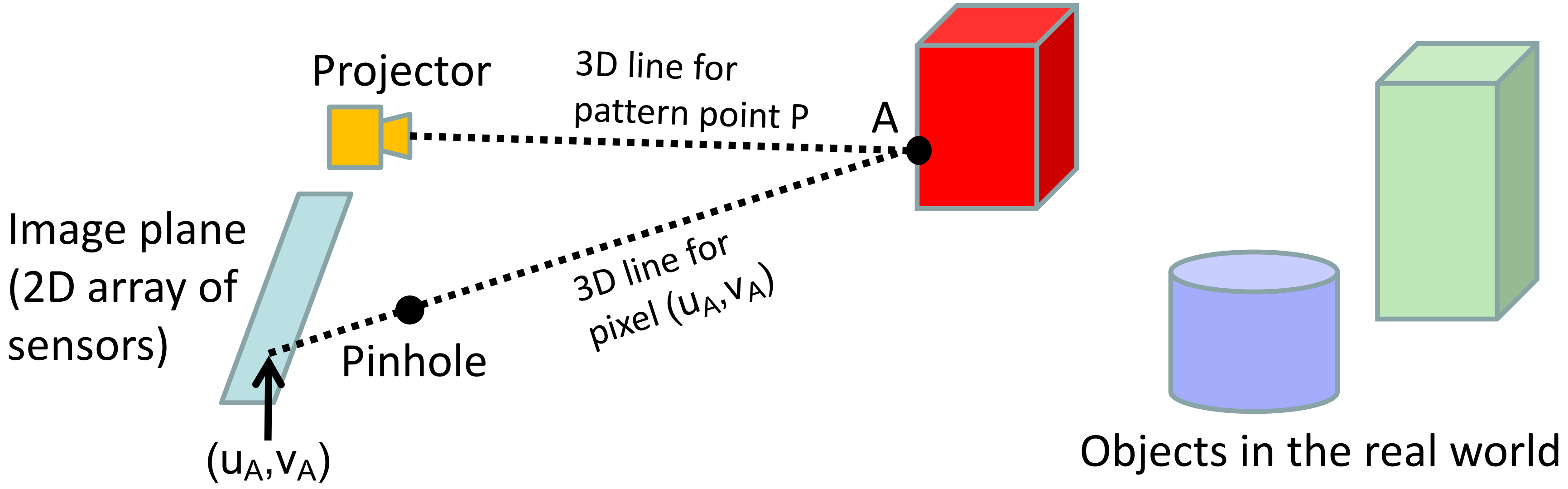
Structured Lighting: Use of Stereo

- The projected pattern is 2-dimensional.
- Every point on the pattern has a 2D pixel location.
- A specific pixel location P on the pattern is projected on a 3D line in the real world.
- This line is known, because the projector is calibrated.


Projector
3D line for pattern point P
A
Image plane (2D array of sensors)
Pinhole
(u_A,v_A)
Objects in the real world

Structured Lighting: Use of Stereo

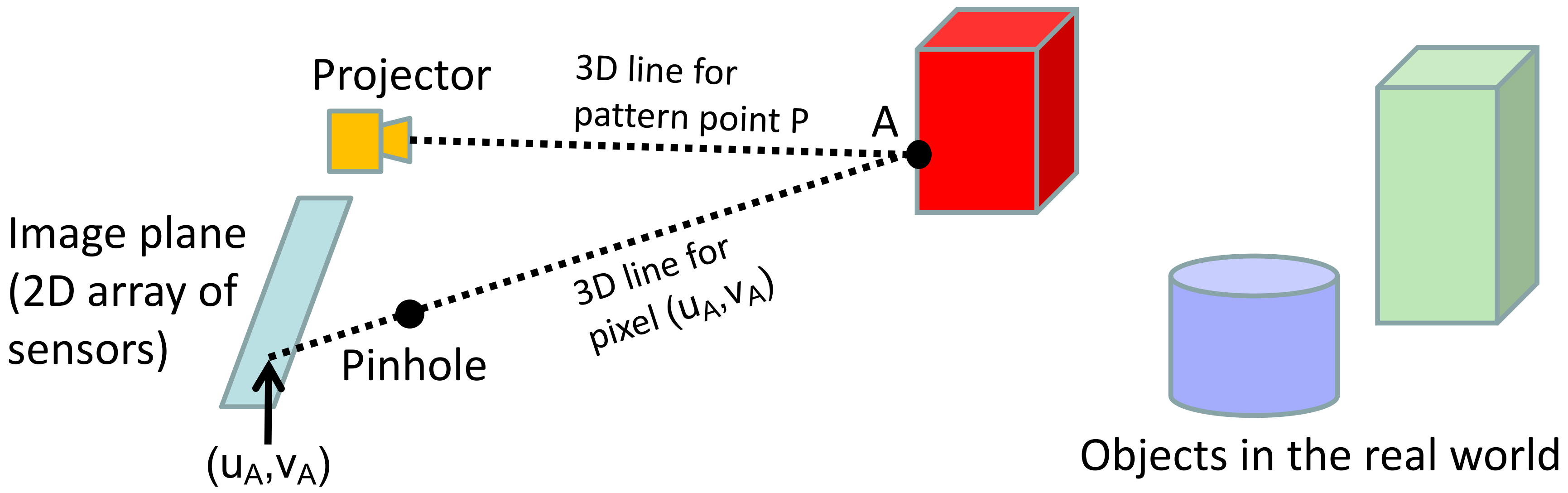
- Suppose that a specific point P on the pattern is detected at pixel location (u_A,v_A) in the image.
- We know that the 3D location of that point is on a specific line, connecting (u_A,v_A) to the camera pinhole.



Projector
3D line for pattern point P
A
Image plane (2D array of sensors)
Pinhole
3D line for pixel (u_A,v_A)
(u_A,v_A)
Objects in the real world

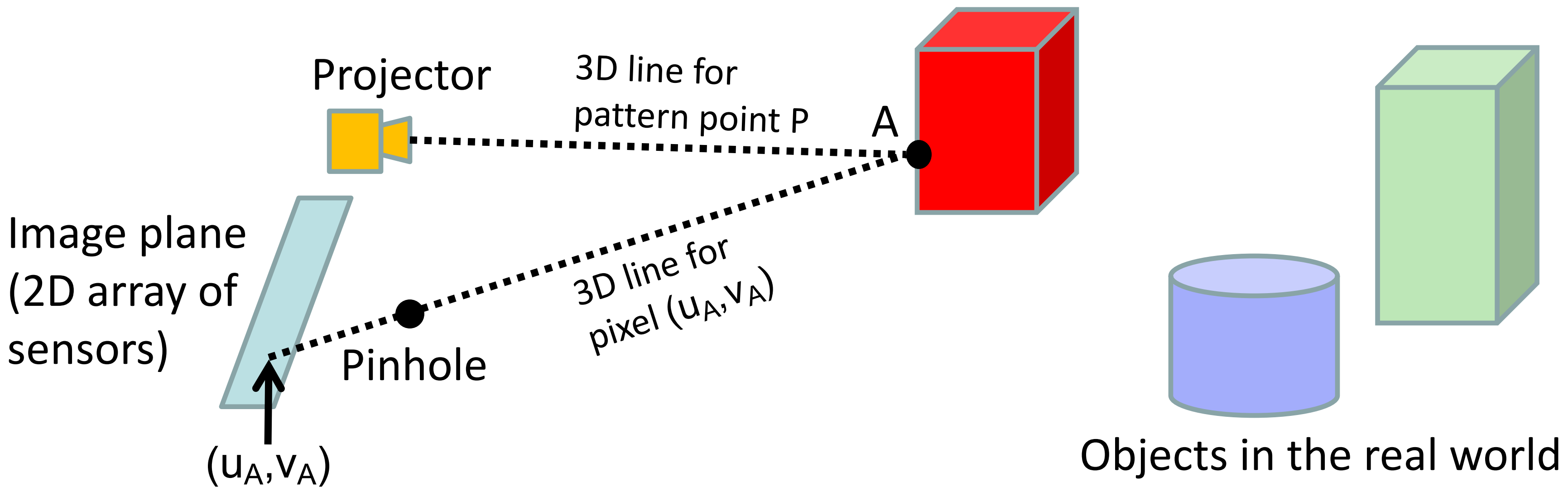
Structured Lighting: Use of Stereo

- So, if pattern point P is detected at pixel (u_A,v_A), we can estimate the corresponding 3D location by finding the “intersection” of two 3D lines:
  - The 3D line corresponding to the pixel location of P in the pattern.
  - The 3D line corresponding to pixel (u_A,v_A) in the image.


Projector
3D line for pattern point P
A
Image plane (2D array of sensors)
Pinhole
3D line for pixel (u_A,v_A)
(u_A,v_A)
Objects in the real world

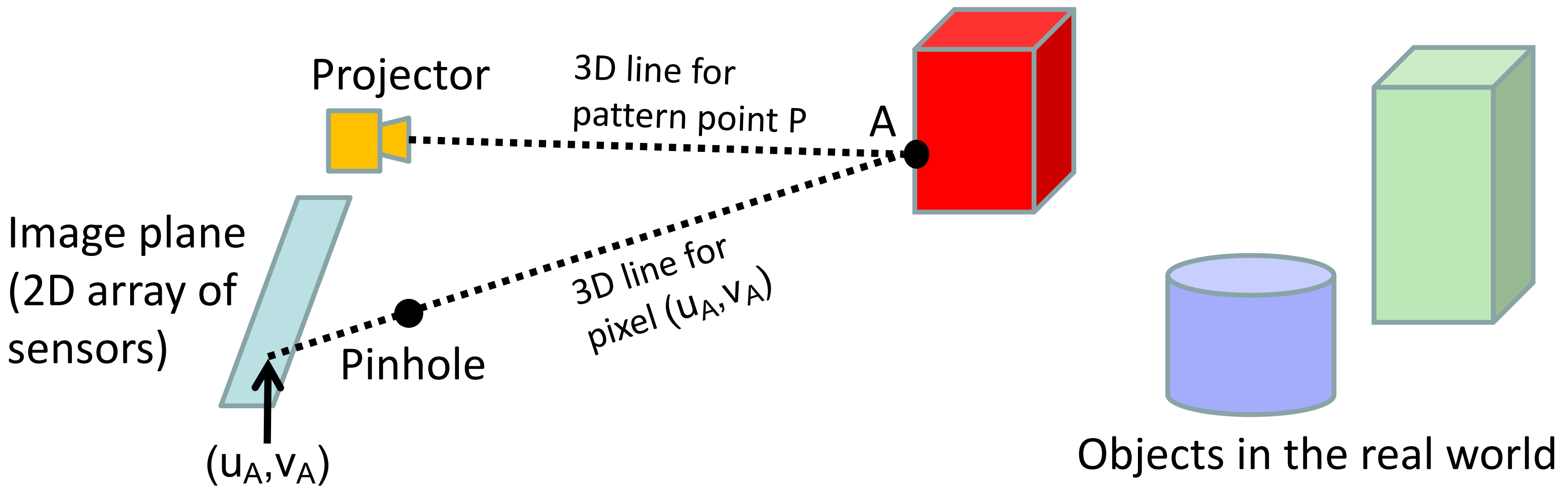
Structured Lighting: Use of Stereo

- The math is similar to the math we use for standard stereo estimation.
- In both cases, we get two 3D lines and we find their intersection.


Projector
3D line for pattern point P
A
Image plane (2D array of sensors)
3D line for pixel (u_A,v_A)
Pinhole
(u_A,v_A)
Objects in the real world

Why Structured Lighting

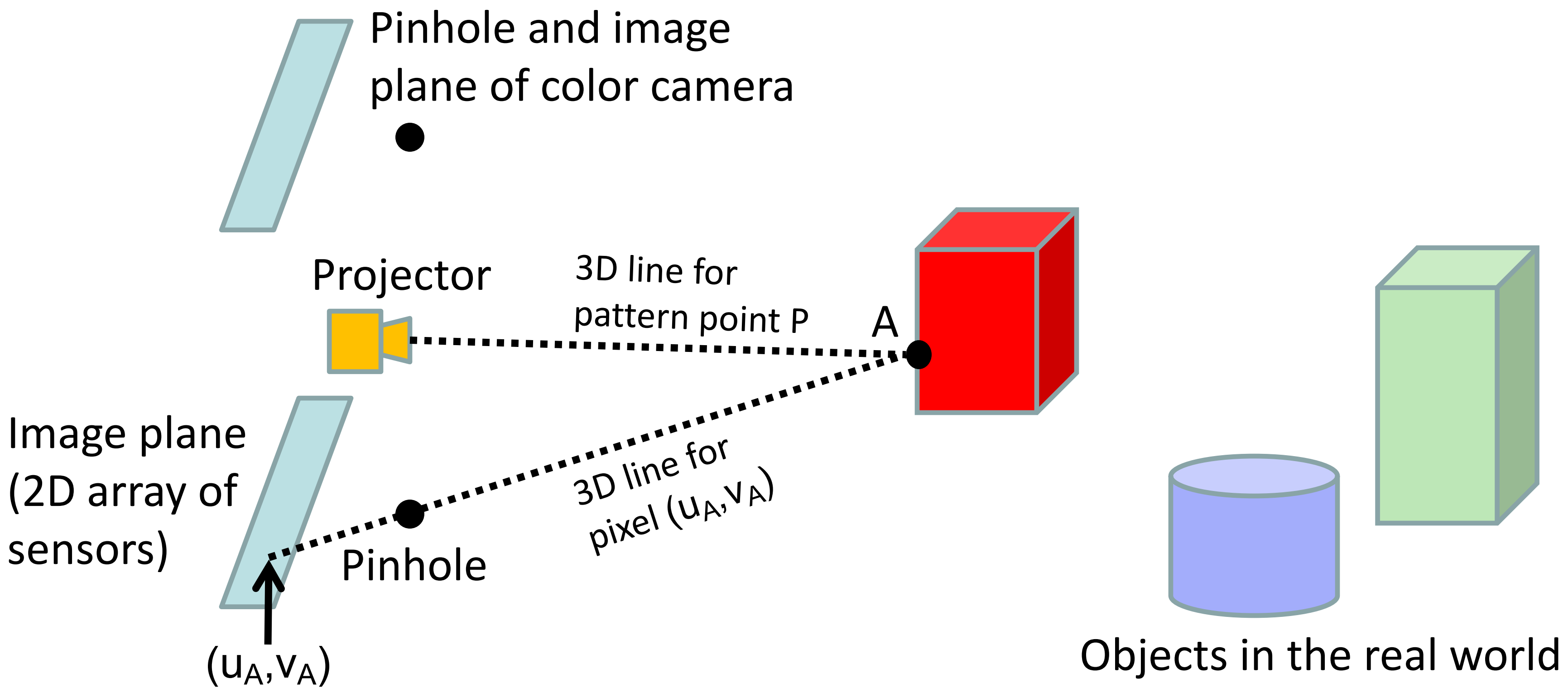
- The main difficulty in standard stereo vision is finding pixel correspondences in the two images.
- In structured lighting, we need to find correspondences between pixels (u_A,v_A) and pattern points P.
- The projected pattern is designed to make it easy to find such correspondences.



Projector
3D line for pattern point P
A
Image plane (2D array of sensors)
Pinhole
3D line for pixel (u_A,v_A)
(u_A,v_A)
Objects in the real world

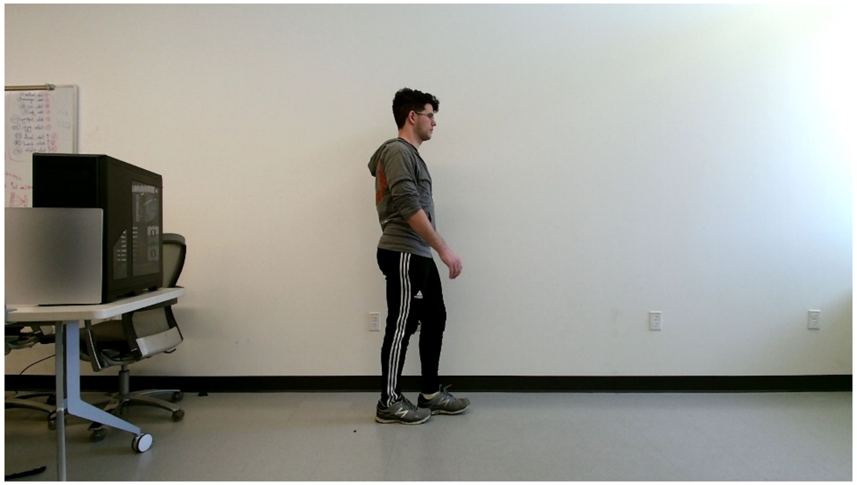
Additional Details

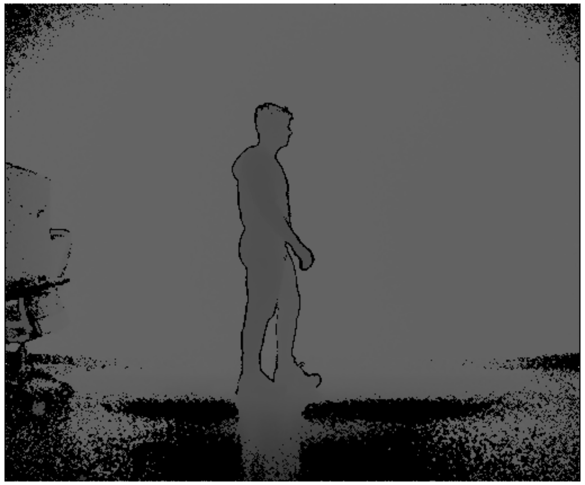
- As in ToF systems, the projected pattern can be infrared, so that it does not bother humans present in the scene.
- As in ToF systems, we usually also include a color camera.


Pinhole and image plane of color camera
Projector
3D line for pattern point P
A
Image plane (2D array of sensors)
Pinhole
3D line for pixel (u_A, v_A)
(u_A, v_A)
Objects in the real world

Failures of Depth Estimation

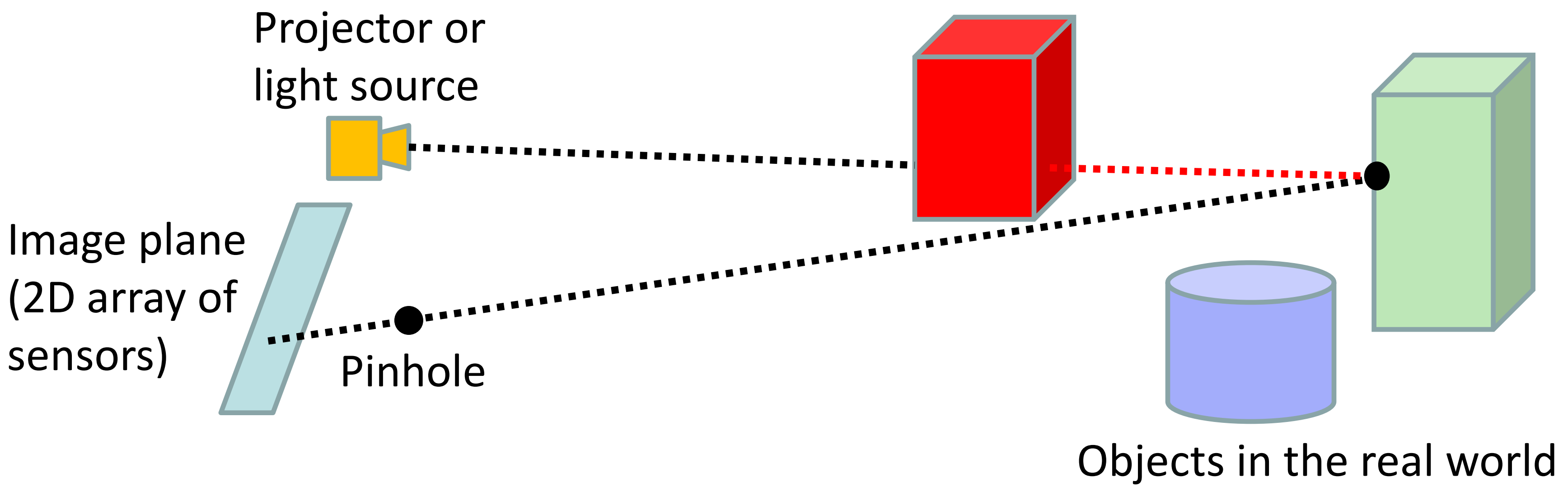
- As we saw in our example, some depth pixels get depth value 0.
- This is a special value meaning “depth estimation failed”.
- Why would depth estimation fail?


color image


depth image

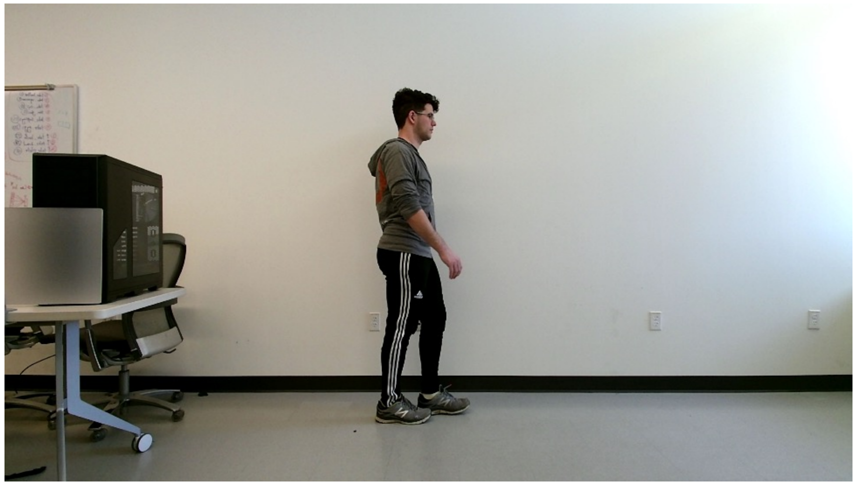
Failures of Depth Estimation

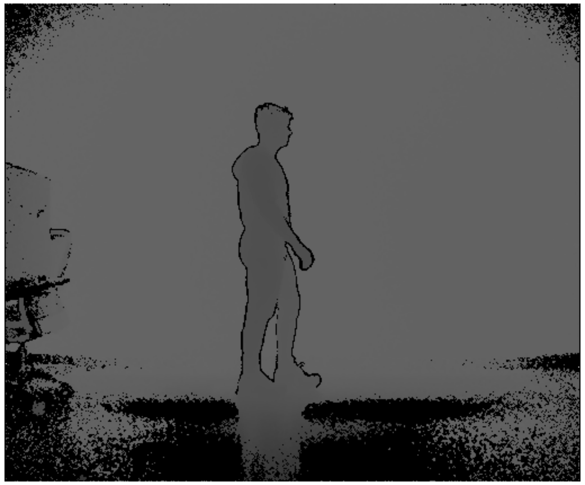
- One reason is that some 3D locations are visible from the camera but not from the projector (for a structured lighting system) or the light source (for a ToF system).
- For such points, there is not enough information to compute depth.


Projector or light source
Image plane (2D array of sensors)
Pinhole
Objects in the real world

Registration of RGB and Depth Images

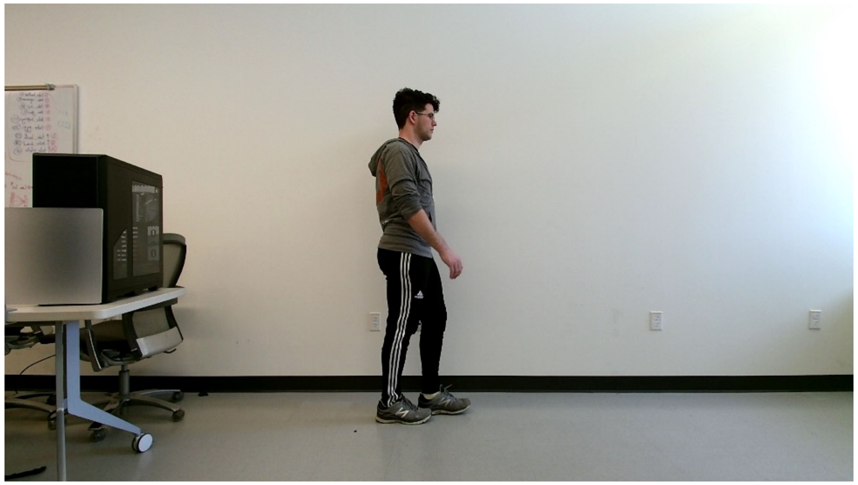
- In principle, for any pixel (u,v) in the color image, we would like to know the depth.
- Unfortunately, depth cameras do not give us such information directly.
- Instead, we get two separate images (color and depth), whose pixels do not correspond to each other.

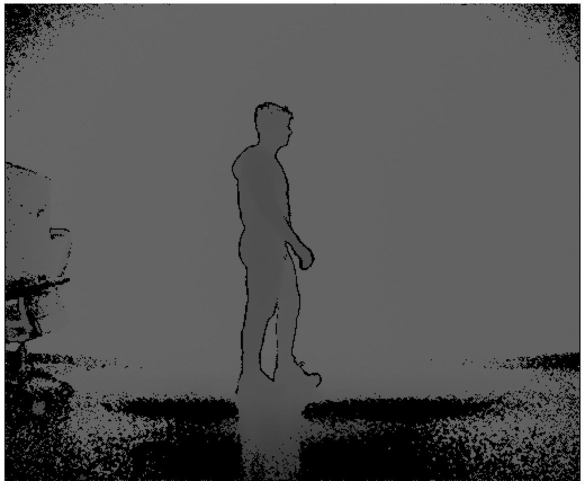

color image


depth image

Registration of RGB and Depth Images

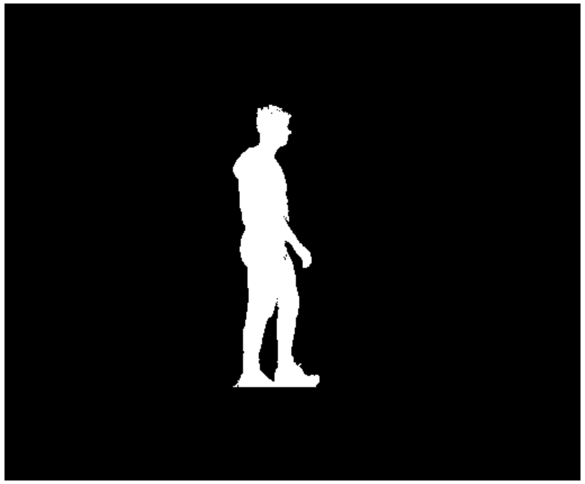
- Identifying the correspondences between color image pixels and depth image pixels is called "registration" of the two images.
- Typically, software libraries packaged with the depth sensor provide functions that do this registration.

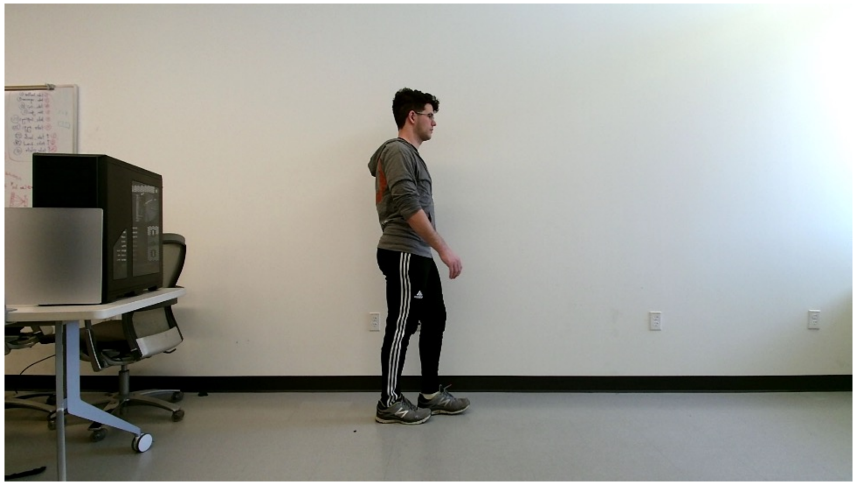

color image

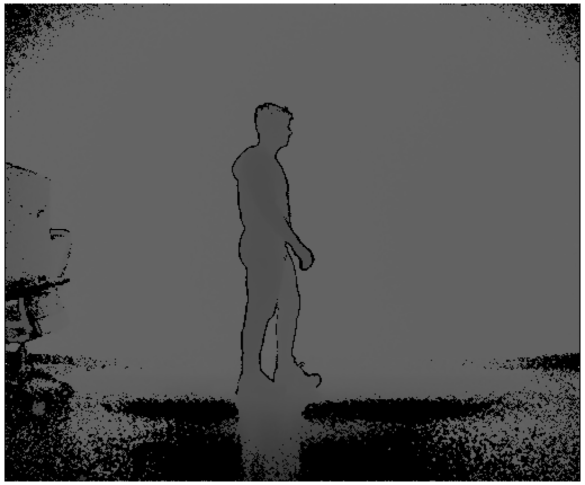

depth image

Depth/Color Registration Results

- Here we can see some example results using a simple camera calibration.
- For every pixel in the person region of the depth image, we put a blue dot on the corresponding pixel in the color image.
  - Correspondences were computed using the process described in the previous slides.

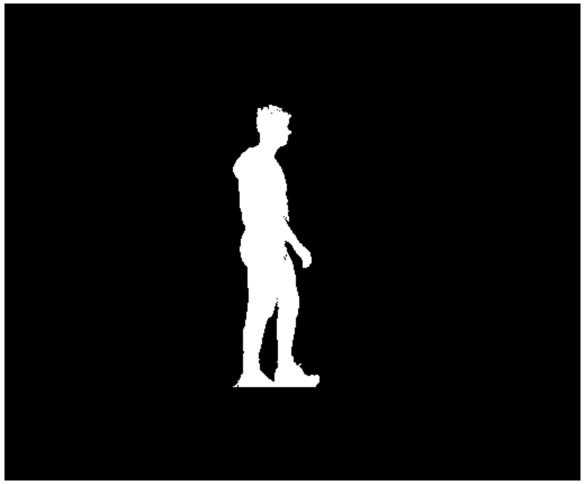

person region

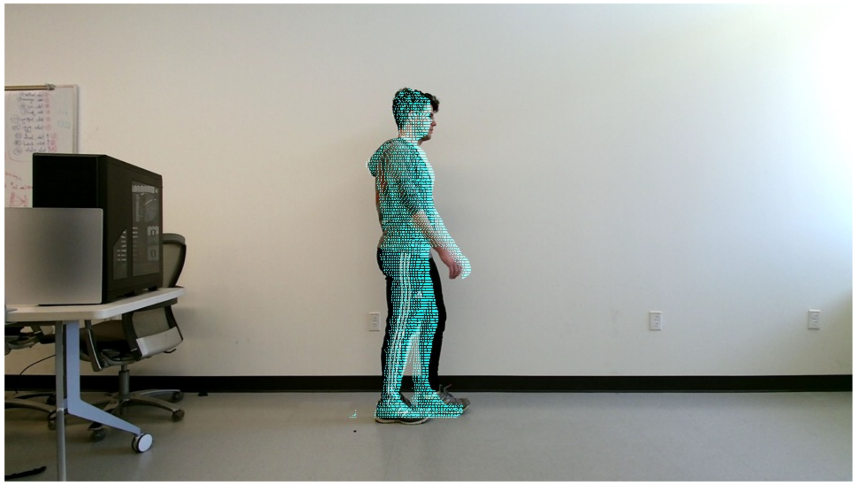

color image

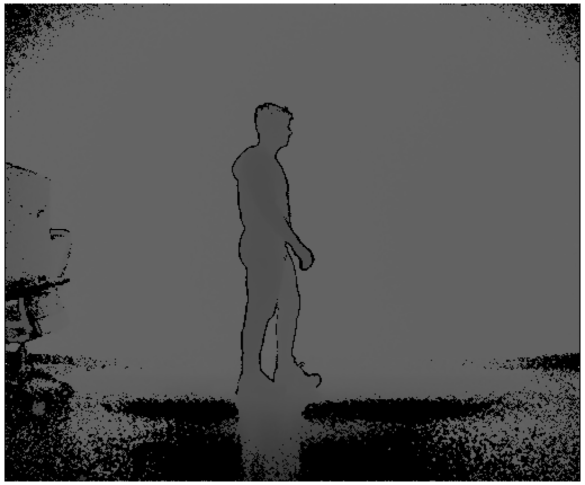

depth image

Depth/Color Registration Results

- Here we can see some example results using a simple camera calibration.
- For every pixel in the person region of the depth image, we put a blue dot on the corresponding pixel in the color image.
- This is what we get.

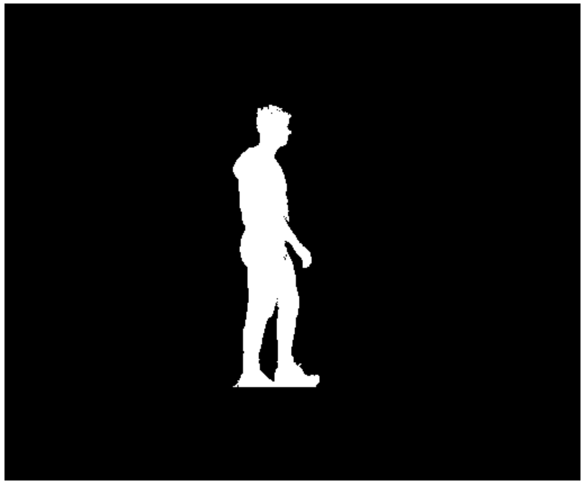

person region

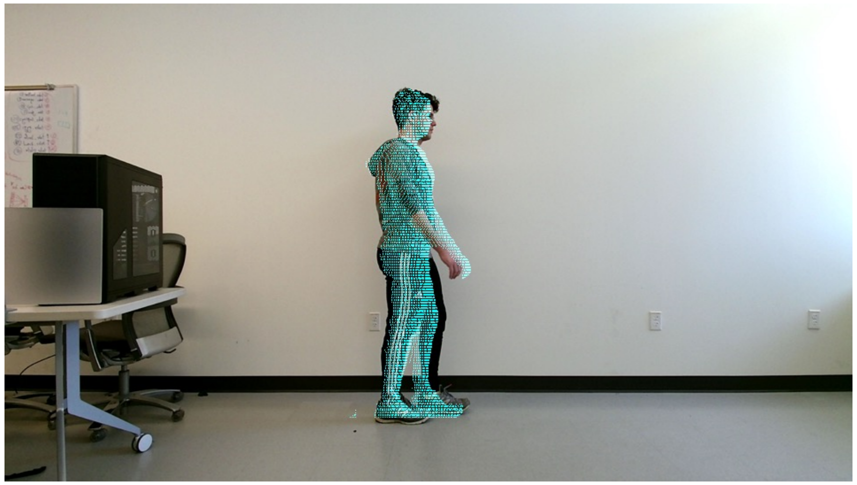

color image

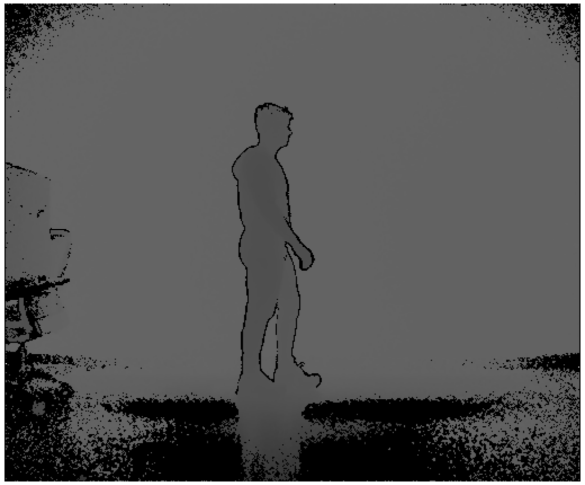

depth image

Depth/Color Registration Results

- The mapping is reasonably close, but not perfect. There are several reasons.
  - The real depth and color cameras do not quite follow the pinhole model.
  - error in the calibration parameters

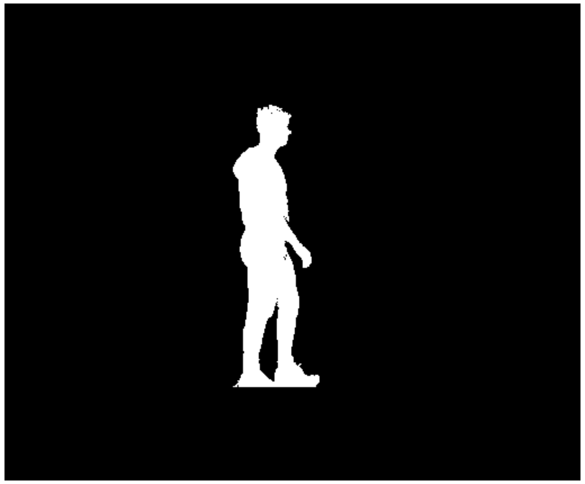

person region

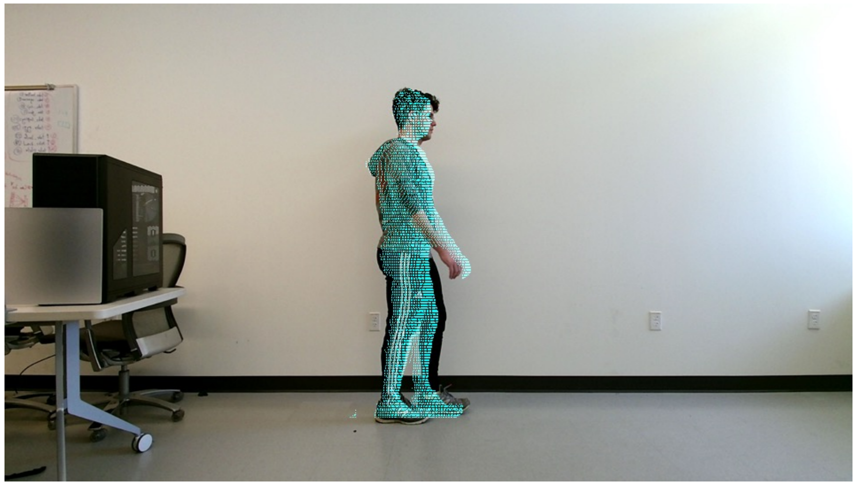

color image

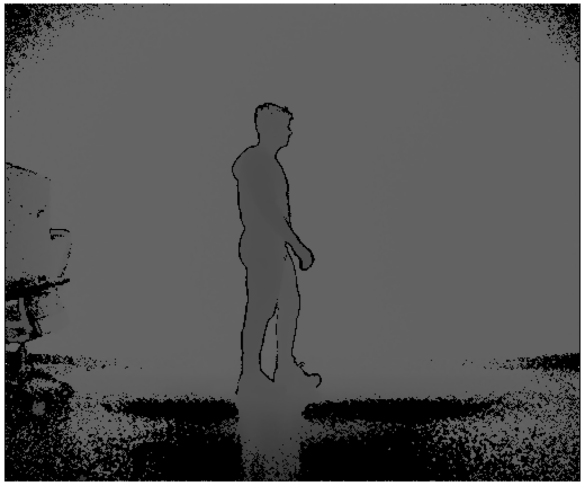

depth image

Depth/Color Registration Results

- More reasons for registration errors:
  - Depth measurements are not 100% accurate.
  - In such cases, where the z value for a depth pixel is wrong, we will not find the correct corresponding pixel in the color image.

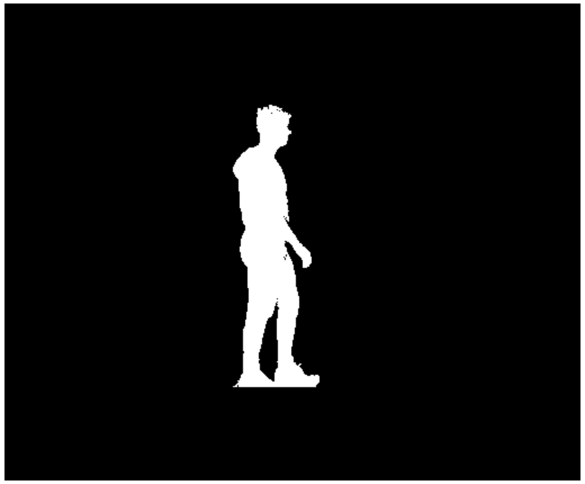

person region

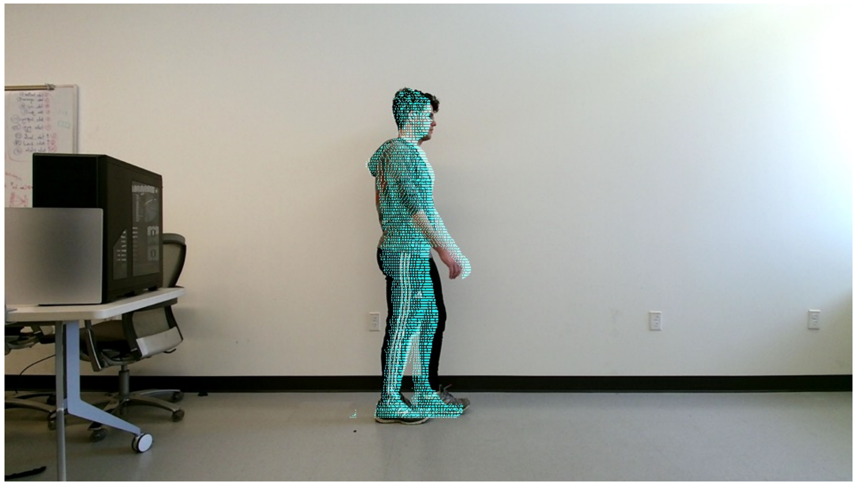

color image

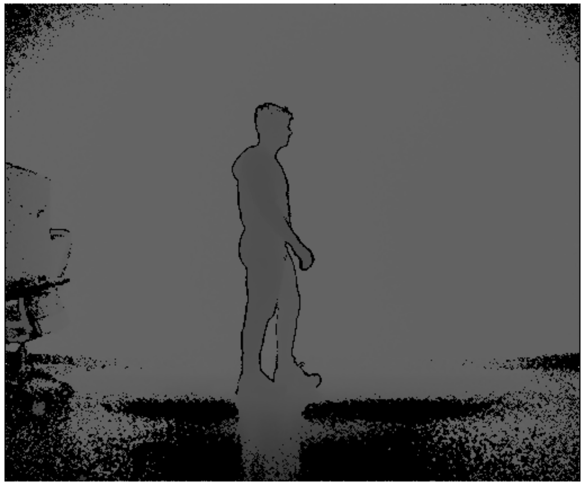

depth image

Depth/Color Registration Results

- More reasons for registration errors:
  - If objects are moving, we need both images to be captured at exactly the same time.
  - In this example, the Kinect camera captured the color image about six milliseconds after the depth image.
  - We can see that the arms and legs moved somewhat during that time.

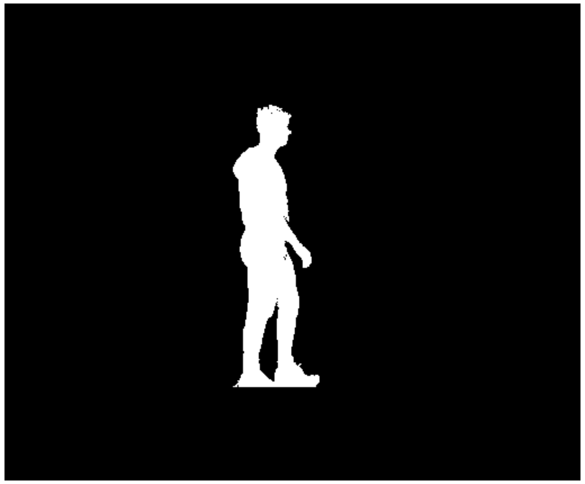

person region

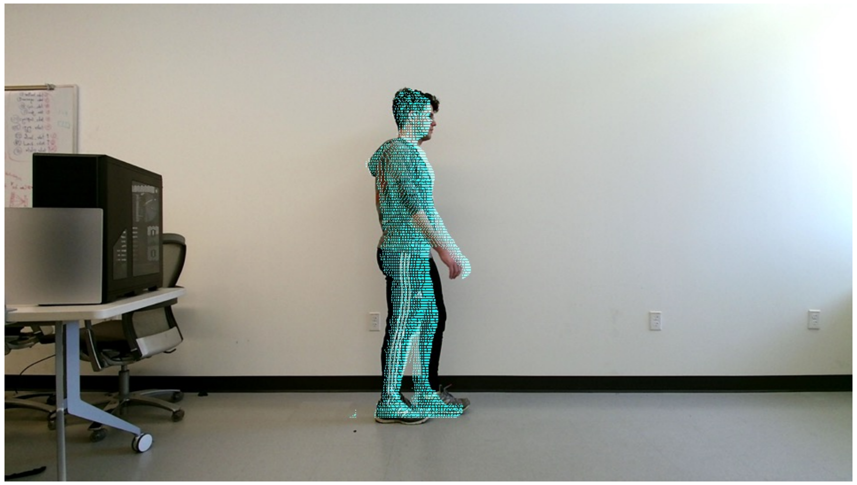

color image

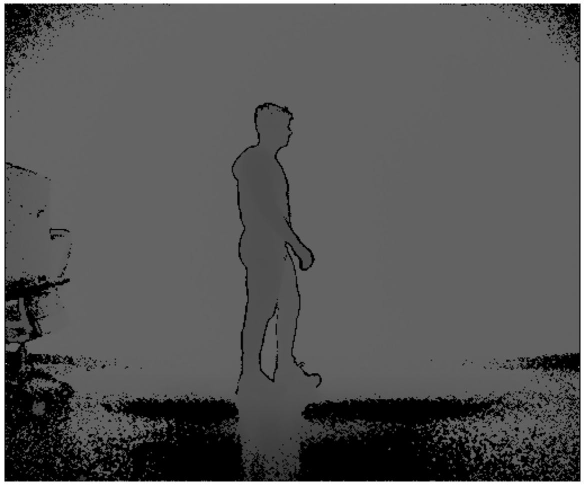

depth image

Depth/Color Registration

- As mentioned earlier, most software packaged with depth sensors provides functions that do this registration automatically.
  - Those functions use more realistic camera models, and produce more accurate results.
  - However, in typical cases the registration is still not perfect and there are noticeable errors.


person region


color image


depth image