# EECE5698
# Networked XR Systems

# Lecture Outline for Today

- Open3D
- Depth Map Compression

# Open3D

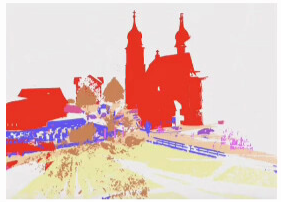
Abstraction Layers

## Applications

Scene reconstruction,
color map
optimization ...

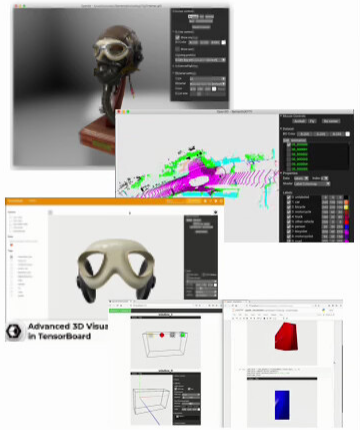
## Open3D-ML

Models: RandLaNet,
KPConv,
PointTransformer, ...
Datasets: SemanticKITTI,
Toronto3D, S3DIS, ...



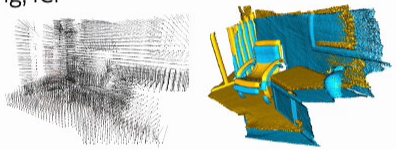
## Visualization

Open3D viewer app,
Scriptable GUI,
Physically-based rendering,
Web visualizer,
TensorBoad integration.



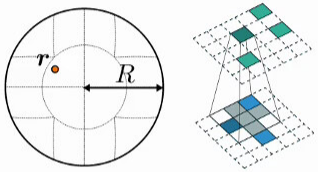

## 3D algorithms

Mesh processing, ICP
Odometry ...



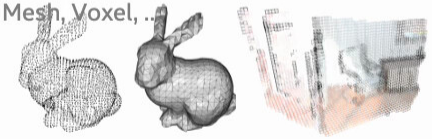
## ML Ops

Continuous conv,
Sparse conv, NMS, ...



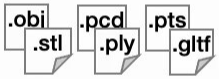

## 3D data structures

Point cloud, Mesh, Voxel, ...



## Sensors & File I/O

Intel RealSense, Azure Kinect, ...
*.pcd, *.ply, *.stl, ...




## Compute core

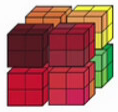
Tensor, HashMap, NeighborSearch
Optimized for CPU/GPU

Devices

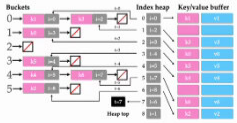

CPU

GPU

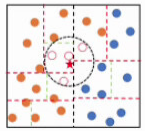
Device-agnostic computer cores



Tensor



HashMap



NeighborSearch

# Open3D

- What will we use Open3D in this source for?
  - Loading and visualizing 3D models
    - Point clouds, Meshes
  - 3D Scene Reconstruction
    - Triangle Extraction
    - Texture Mapping
  - Mesh manipulation
    - Decimation
    - Normal estimation
    - Compression

# Open3D Getting Started

- Python quick start

```
# Install
pip install open3d       # or
pip install open3d-cpu   # Smaller CPU only wheel on x86_64 Linux (v0.17+)

# Verify installation
python -c "import open3d as o3d; print(o3d.__version__)"
```

# Open3D Getting Started

- C++ quick start

Checkout the following links to get started with Open3D C++ API

- Download Open3D binary package: Release or latest development version
- Compiling Open3D from source
- Open3D C++ API

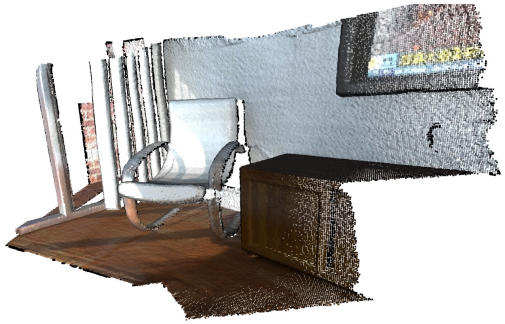
To use Open3D in your C++ project, checkout the following examples

- Find Pre-Installed Open3D Package in CMake
- Use Open3D as a CMake External Project

# Open3D Point Cloud Visualization

```
print("Load a ply point cloud, print it, and render it")
ply_point_cloud = o3d.data.PLYPointCloud()
pcd = o3d.io.read_point_cloud(ply_point_cloud.path)
print(pcd)
print(np.asarray(pcd.points))
o3d.visualization.draw_geometries([pcd],
                                  zoom=0.3412,
                                  front=[0.4257, -0.2125, -0.8795],
                                  lookat=[2.6172, 2.0475, 1.532],
                                  up=[-0.0694, -0.9768, 0.2024])
```

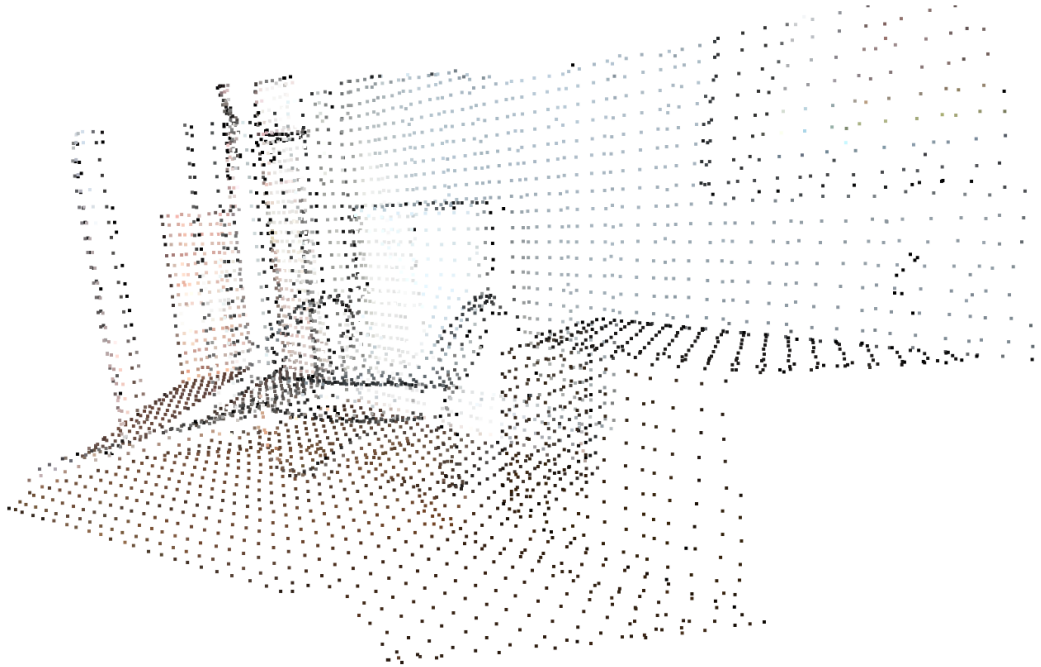
```
Load a ply point cloud, print it, and render it
PointCloud with 196133 points.
[[0.65234375 0.84686458 2.37890625]
 [0.65234375 0.83984375 2.38430572]
 [0.66737998 0.83984375 2.37890625]
 ...
 [2.00839925 2.39453125 1.88671875]
 [2.00390625 2.39488506 1.88671875]
 [2.00390625 2.39453125 1.88793314]]
```



# Open3D Point Cloud Visualization

```
print("Downsample the point cloud with a voxel of 0.05")
downpcd = pcd.voxel_down_sample(voxel_size=0.05)
o3d.visualization.draw_geometries([downpcd],
                                  zoom=0.3412,
                                  front=[0.4257, -0.2125, -0.8795],
                                  lookat=[2.6172, 2.0475, 1.532],
                                  up=[-0.0694, -0.9768, 0.2024])
```

```
Downsample the point cloud with a voxel of 0.05
```



Down sampling

# Open3D Mesh Visualization

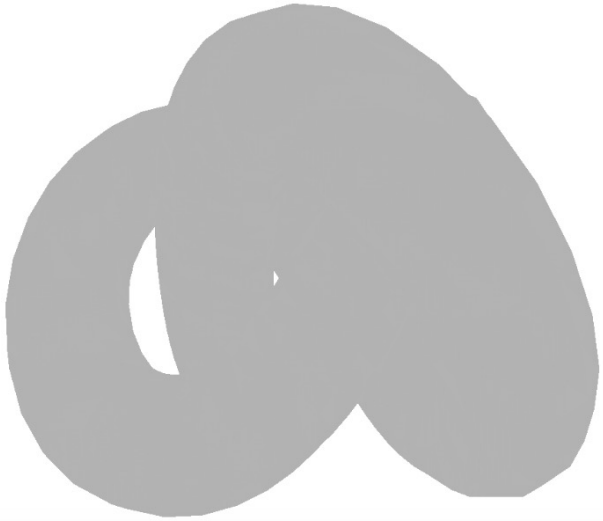
```
print("Testing mesh in Open3D...")
armadillo_mesh = o3d.data.ArmadilloMesh()
mesh = o3d.io.read_triangle_mesh(armadillo_mesh.path)

knot_mesh = o3d.data.KnotMesh()
mesh = o3d.io.read_triangle_mesh(knot_mesh.path)
print(mesh)
print('Vertices:')
print(np.asarray(mesh.vertices))
print('Triangles:')
print(np.asarray(mesh.triangles))
```

```
Testing mesh in Open3D...
[Open3D INFO] Downloading https://github.com/isl-org/open3d_downloads/releases/download/
[Open3D INFO] Downloaded to /home/runner/open3d_data/download/KnotMesh/KnotMesh.ply
TriangleMesh with 1440 points and 2880 triangles.
Vertices:
[[  4.51268387  28.68865967 -76.55680847]
 [  7.63622284  35.52046967 -69.78063965]
 [  6.21986008  44.22465134 -64.82303619]
 ...
 [-22.12651634  31.28466606 -87.37570953]
 [-13.91188431  25.4865818  -86.25827026]
 [ -5.27768707  23.36245346 -81.43279266]]
Triangles:
[[   0   12   13]
 [   0   13    1]
 [   1   13   14]
 ...
 [1438   11 1439]
 [1439   11    0]
 [1439    0 1428]]
```

# Open3D Mesh Visualization

```
print("Try to render a mesh with normals (exist: " +
      str(mesh.has_vertex_normals()) + ") and colors (exist: " +
      str(mesh.has_vertex_colors()) + ")")
o3d.visualization.draw_geometries([mesh])
print("A mesh with no normals and no colors does not look good.")
```

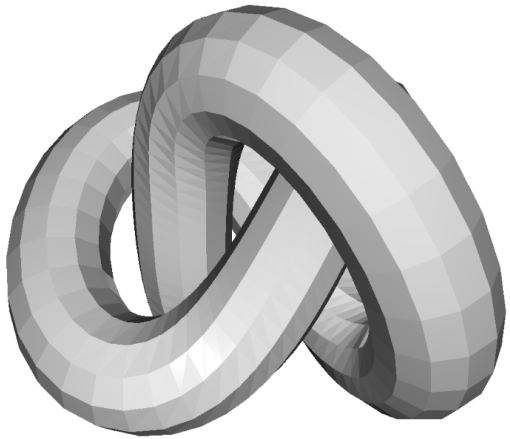
```
Try to render a mesh with normals (exist: False) and colors (exist: False)
```



# Open3D Mesh Visualization

```
print("Computing normal and rendering it.")
mesh.compute_vertex_normals()
print(np.asarray(mesh.triangle_normals))
o3d.visualization.draw_geometries([mesh])
```

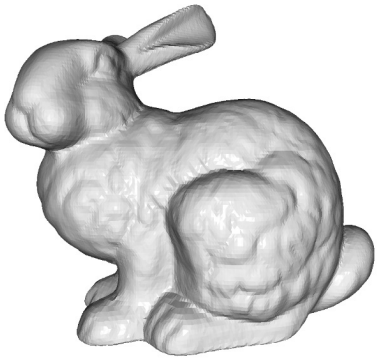
```
Computing normal and rendering it.
[[ 0.79164373 -0.53951444  0.28674793]
 [ 0.8319824  -0.53303008  0.15389681]
 [ 0.83488162 -0.09250101  0.54260136]
 ...
 [ 0.16269924 -0.76215917 -0.6266118 ]
 [ 0.52755226 -0.83707495 -0.14489352]
 [ 0.56778973 -0.76467734 -0.30476777]]
```

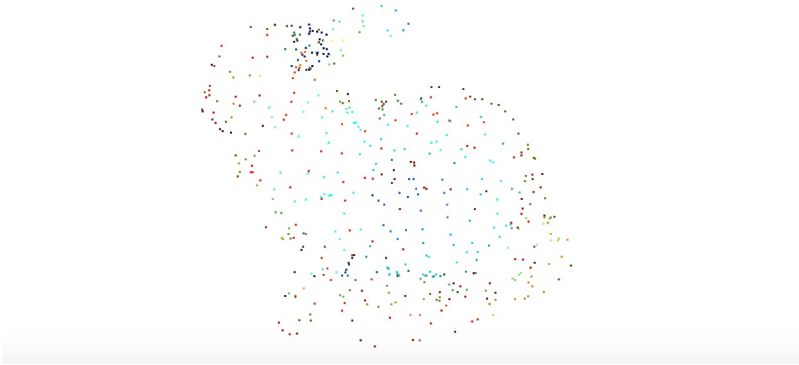


With Normals

# Open3D Mesh Visualization

```
bunny = o3d.data.BunnyMesh()
mesh = o3d.io.read_triangle_mesh(bunny.path)
mesh.compute_vertex_normals()

o3d.visualization.draw_geometries([mesh])
pcd = mesh.sample_points_uniformly(number_of_points=500)
o3d.visualization.draw_geometries([pcd])
```

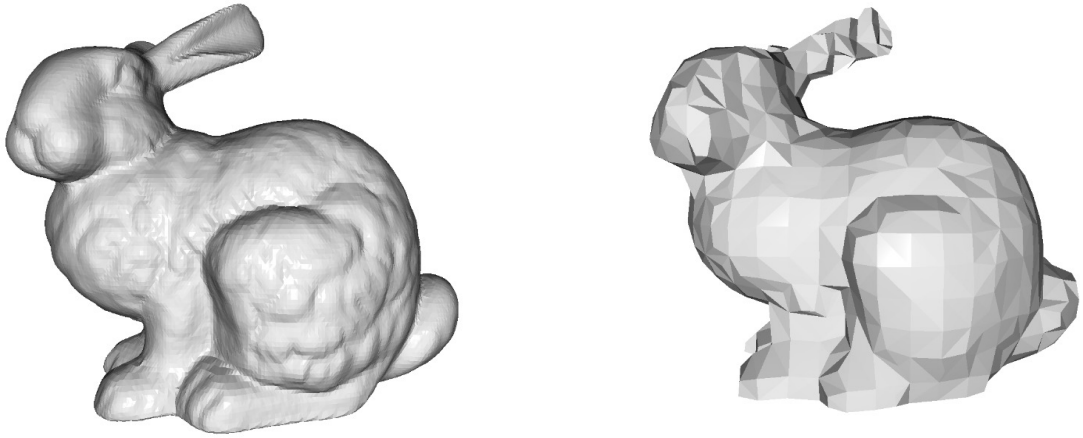




Sampling Point Clouds from Mesh

# Open3D Mesh Visualization

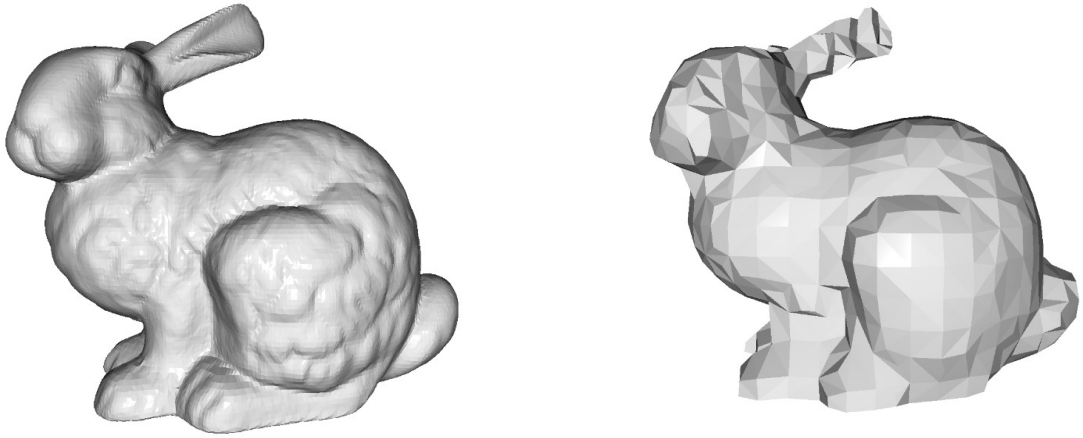
```
voxel_size = max(mesh_in.get_max_bound() - mesh_in.get_min_bound()) / 32
print(f'voxel_size = {voxel_size:e}')
mesh_smp = mesh_in.simplify_vertex_clustering(
    voxel_size=voxel_size,
    contraction=o3d.geometry.SimplificationContraction.Average)
print(
    f'Simplified mesh has {len(mesh_smp.vertices)} vertices and {len(mesh_smp.triangles)}
)
o3d.visualization.draw_geometries([mesh_smp])
```



Mesh Simplification – Vertex clustering

# Open3D Mesh Visualization

```
mesh_smp = mesh_in.simplify_quadric_decimation(target_number_of_triangles=6500)
print(
    f'Simplified mesh has {len(mesh_smp.vertices)} vertices and {len(mesh_smp.triangles)}
)
o3d.visualization.draw_geometries([mesh_smp])

mesh_smp = mesh_in.simplify_quadric_decimation(target_number_of_triangles=1700)
print(
    f'Simplified mesh has {len(mesh_smp.vertices)} vertices and {len(mesh_smp.triangles)}
)
o3d.visualization.draw_geometries([mesh_smp])
```



Mesh Simplification – Decimation

# Open3D Mesh Reconstruction

```
print('run Poisson surface reconstruction')
with o3d.utility.VerbosityContextManager(
        o3d.utility.VerbosityLevel.Debug) as cm:
    mesh, densities = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(
        pcd, depth=9)
print(mesh)
o3d.visualization.draw_geometries([mesh],
                                  zoom=0.664,
                                  front=[-0.4761, -0.4698, -0.7434],
                                  lookat=[1.8900, 3.2596, 0.9284],
                                  up=[0.2304, -0.8825, 0.4101])
```

Read a point cloud or depth before this function

# Open3D Load and Save Viewpoints

```python
def save_view_point(pcd, filename):
    vis = o3d.visualization.Visualizer()
    vis.create_window()
    vis.add_geometry(pcd)
    vis.run()  # user changes the view and press "q" to terminate
    param = vis.get_view_control().convert_to_pinhole_camera_parameters()
    o3d.io.write_pinhole_camera_parameters(filename, param)
    vis.destroy_window()


def load_view_point(pcd, filename):
    vis = o3d.visualization.Visualizer()
    vis.create_window()
    ctr = vis.get_view_control()
    param = o3d.io.read_pinhole_camera_parameters(filename)
    vis.add_geometry(pcd)
    ctr.convert_from_pinhole_camera_parameters(param)
    vis.run()
    vis.destroy_window()
```

# Open3D UV Maps

## What is a UV Map ?

[Ref: Wikipedia] UV mapping is the 3D modeling process of projecting a 2D image to a 3D model's surface for texture mapping. The letters "U" and "V" denote the axes of the 2D texture because "X", "Y", and "Z" are already used to denote the axes of the 3D object in model space. UV texturing permits polygons that make up a 3D object to be painted with color (and other surface attributes) from an ordinary image. The image is called a UV texture map.

## How to add custom UV maps ? #

- UV Map coordinates `(U, V)` are stored as `std::vector<Eigen::Vector2d>` of length `3 x number of triangles`. So, there is a set of 3 (U, V) coordinates for each triangle, each associated with it's vertices.
- One may assume the UV map, maps a texture image of height and width of length 1.0 to the geometry. So, the range of U and V is from 0.0 to 1.0 (both inclusive).

# Open3D UV Maps

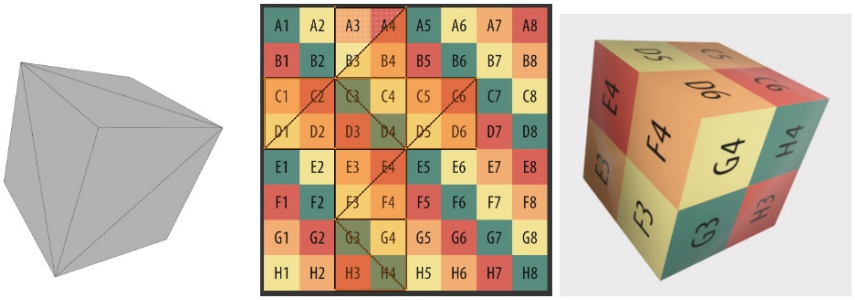
```
import open3d as o3d
import open3d.visualization.rendering as rendering

material = rendering.MaterialRecord()
material.shader = 'defaultUnlit'
material.albedo_img = o3d.io.read_image('/Users/renes/Downloads/uv1.png')
```

## Example Texture Map

# Open3D UV Maps

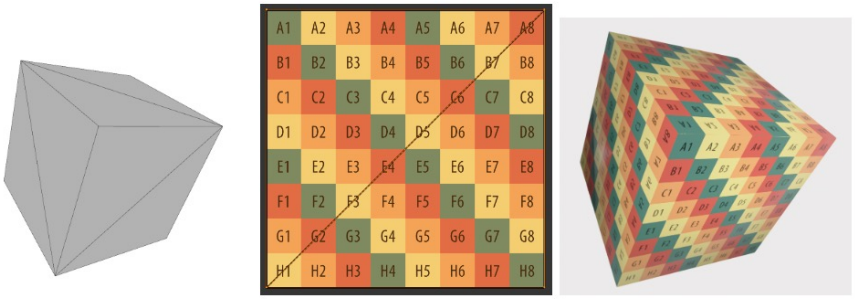
### Box (map uv to each face = false) #

```
box = o3d.geometry.TriangleMesh.create_box(create_uv_map=True)
o3d.visualization.draw({'name': 'box', 'geometry': box, 'material': material})
```




### Box (map uv to each face = true)

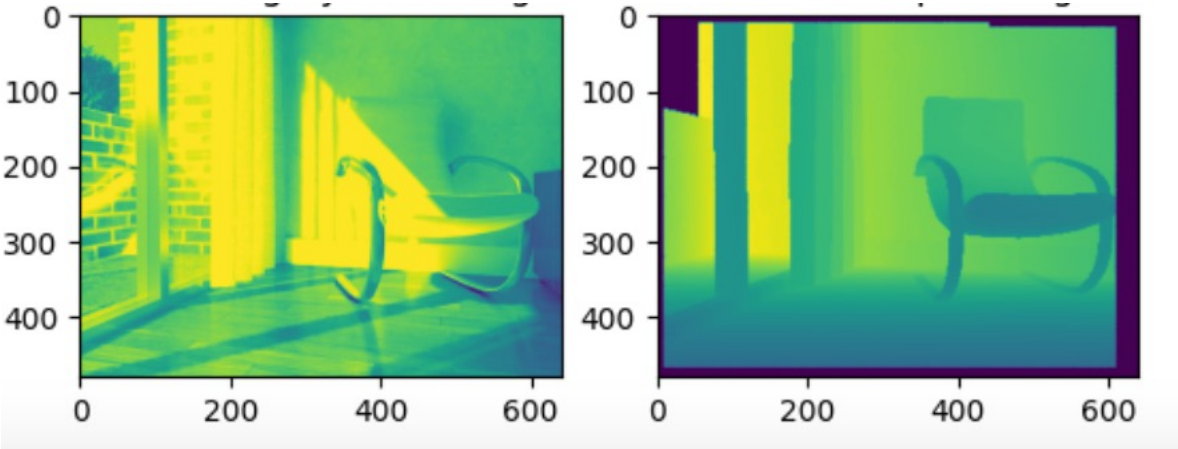
```
box = o3d.geometry.TriangleMesh.create_box(create_uv_map=True, map_texture_to_each_face=True
o3d.visualization.draw({'name': 'box', 'geometry': box, 'material': material})
```



# Open3D Depth Map Viewer

```
print("Read Redwood dataset")
redwood_rgbd = o3d.data.SampleRedwoodRGBDImages()
color_raw = o3d.io.read_image(redwood_rgbd.color_paths[0])
depth_raw = o3d.io.read_image(redwood_rgbd.depth_paths[0])
rgbd_image = o3d.geometry.RGBDImage.create_from_color_and_depth(
    color_raw, depth_raw)
print(rgbd_image)
```

```
plt.subplot(1, 2, 1)
plt.title('Redwood grayscale image')
plt.imshow(rgbd_image.color)
plt.subplot(1, 2, 2)
plt.title('Redwood depth image')
plt.imshow(rgbd_image.depth)
plt.show()
```

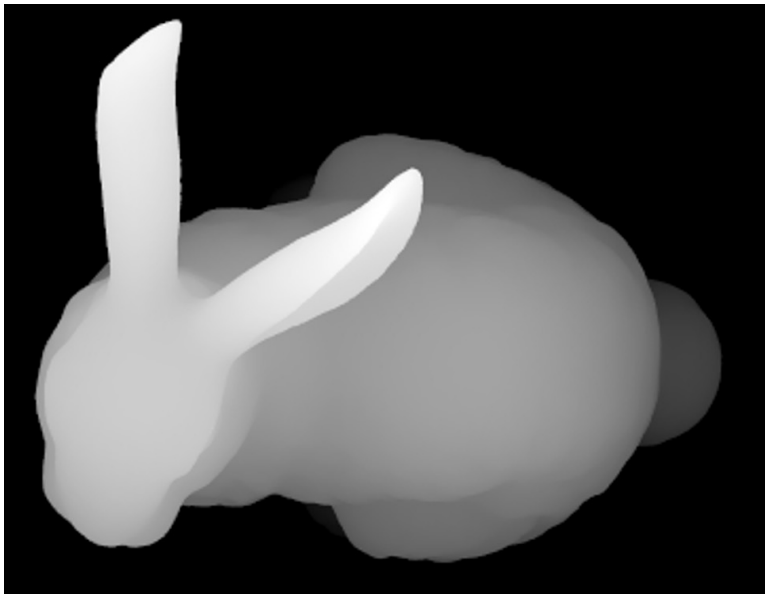


# Depth Map Compression

- Why do we need depth compression?
  - For storage
  - For streaming over a network under lower bandwidths
  - For 3D scene reconstruction on client devices

# Depth Map Compression

- How to compress depth?
  - Can we use similar ideas that we discussed in case of 2D video codecs?
  - Frame prediction
  - Transform coding & quantization
  - Entropy coding



# Lecture Outline for Today

- Open3D
- Depth Map Compression

# Depth Map Compression

- Why not just adopt standard video codecs?
  - They have been engineered for decades
  - Probably no need to reinvent similar algorithms if we can directly input the depth videos to color video codecs

# Adopting Standard Video Codecs

- Challenge
  - Compression schemes for standard videos are highly tuned for color videos
  - Considering human perception, e.g., by spending fewer bits on color than luminance information, and so forth.
  - These insights do not apply to depth compression.

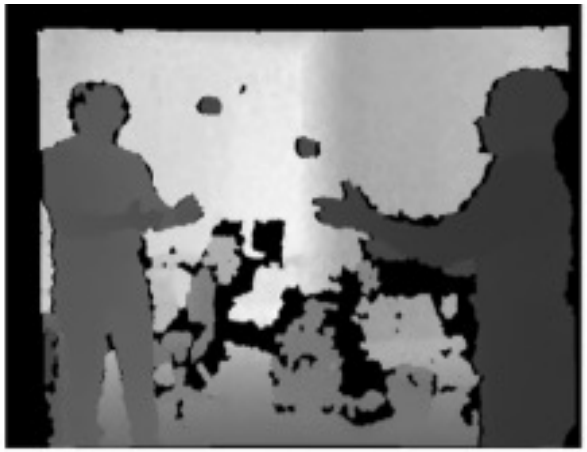
# Adopting Standard Video Codecs

- Challenge
  - Bit-depth and channel inconsistency
  - Depth videos are single channeled
  - Bit-depth of depth videos is larger than color videos in general

  - Example: 8-bit videos can only store coarse-grained depth or short range (trade-off), so typically you need 16-bit or 24-bit or 32-bit-detphs for depth videos

# Adopting Standard Video Codecs

- Can we convert the single channel large bit-depths to three channel small bit-depth?



https://jankautz.com/publications/depth-streaming.pdf

# Adopting Standard Video Codecs

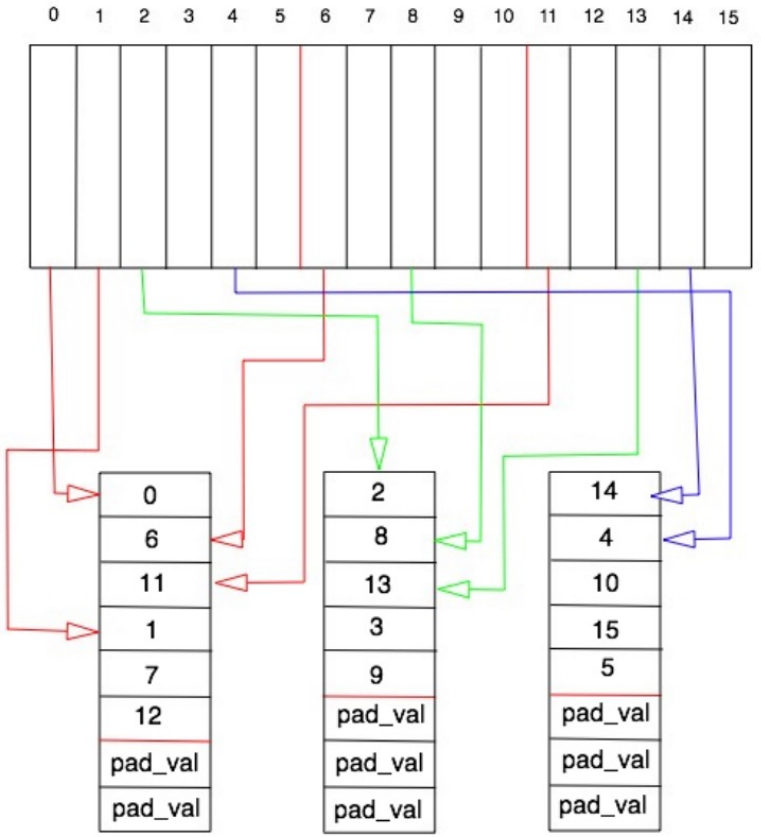
- Example: How do we pack a single 16 bit-depth depth map into a three channel 8 bit-depth depth map?
- Bit Multiplexing Method
  - Take a chunk of bits from 16 bits, insert them in each in 8-bit channel
  - E.g., first 6 MSB of 16-bits into first 6 MSB of first channel, subsequent 5 bits into first 5 MSB of second channel, last 5 bits into first 5 MSB of third channel
  - Pad with zeros for the remaining bits

# Adopting Standard Video Codecs

- Example: How do we pack a single 16 bit-depth depth map into a three channel 8 bit-depth depth map?
- Bit Multiplexing Method
  - Take a chunk of bits from 16 bits, insert them in each in 8-bit channel
  - Problems?
    - Loss in MSBs can cause large depth discontinuities

# Adopting Standard Video Codecs

- Example: How do we pack a single 16 bit-depth depth map into a three channel 8 bit-depth depth map?
- Interleaved Bit Multiplexing
  - Slightly better solution




https://jankautz.com/publications/depth-streaming.pdf

# Adapting Standard Video Codecs

- General Limitations
  - Lossy
- Lossless entropy coding
    - Fast Lossless Depth Image Compression, ISS'17
      - Skips frame prediction, transform and quantization to avoid depth discontinuities
      - Applies entropy coding
      - But only for images, not for video
    - Temporal RVL: A Depth Stream Compression Method, IEEE VR'20
      - For videos; computes deltas across frames

# Lecture Summary

- Open3D
- Depth Map Compression

Next up: Point Cloud Compression