EECE5512
Networked XR Systems

Last Class - Recap

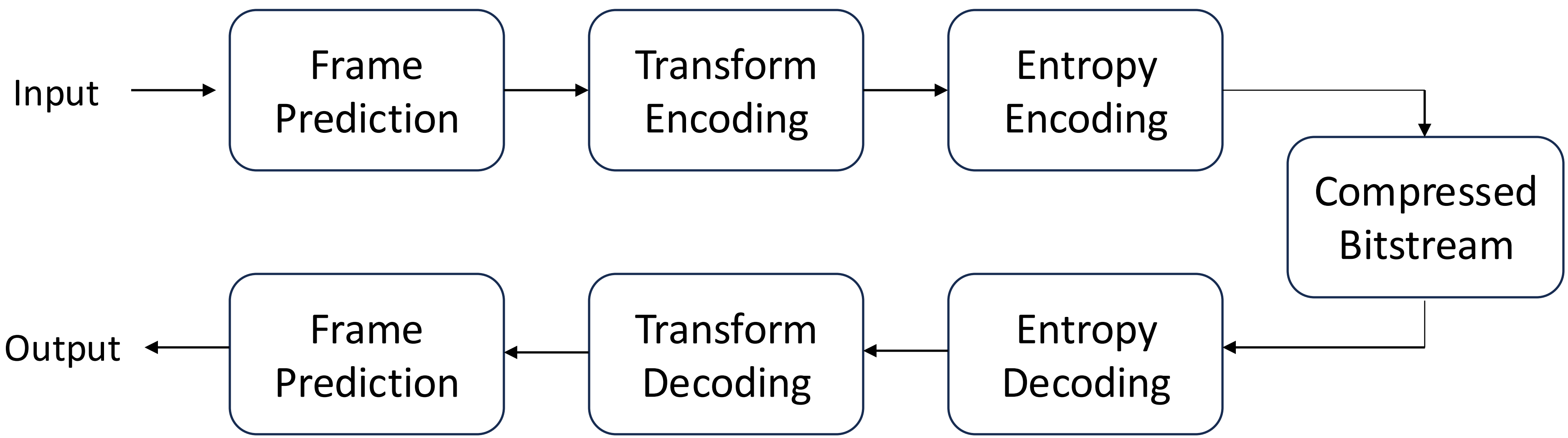
- Live 3D Capture
- Network Capacity vs. Requirements of Applications
- Compression Fundamentals
- 2D Video Compression

Lecture Outline for Today

- 2D Video Compression
- Open3D
- Depth Map Compression

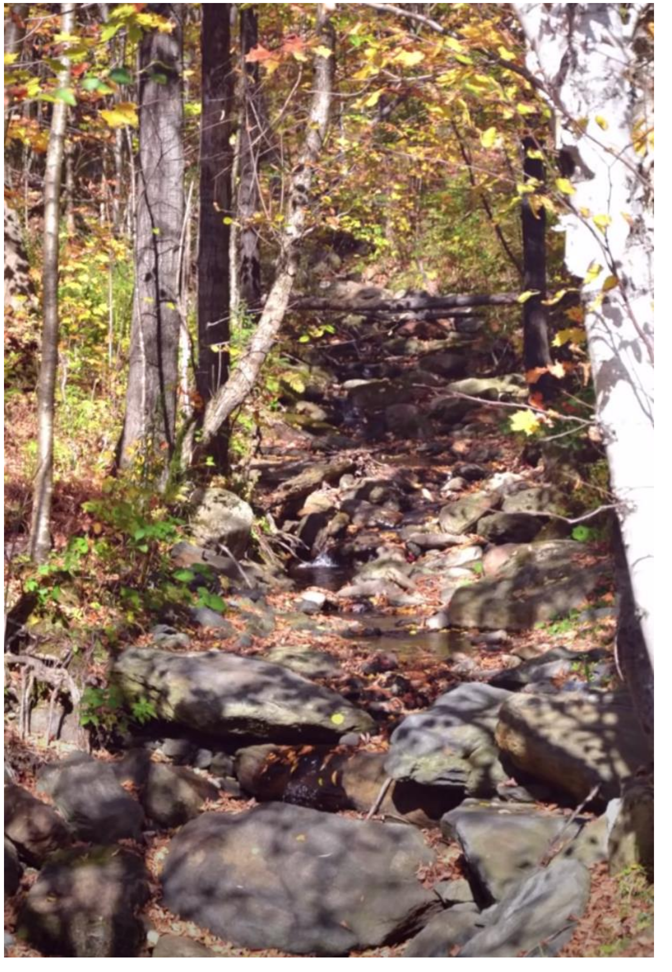
Compression Fundamentals

- Key steps involved in video compression pipeline
  - Color space or Chroma sub-sampling



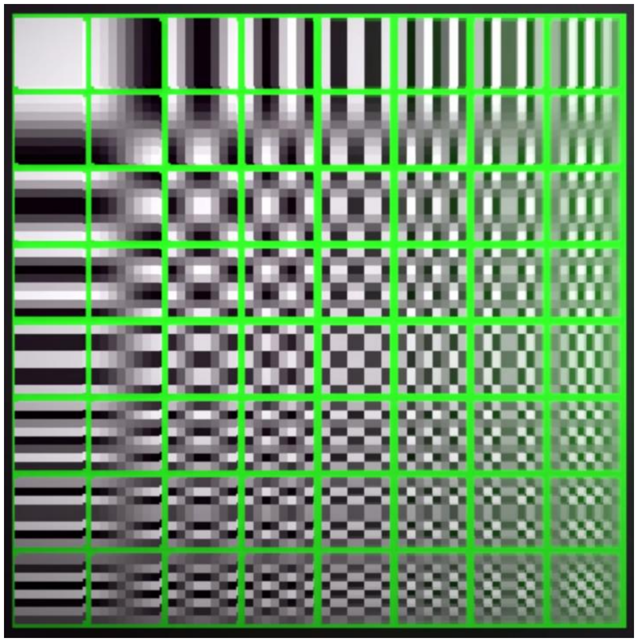
Transform Coding & Quantization

- Transform encoding and quantization
  - Our eyes are bad at perceiving high frequency data
  - Throw away a lot of such data – negligible quality loss



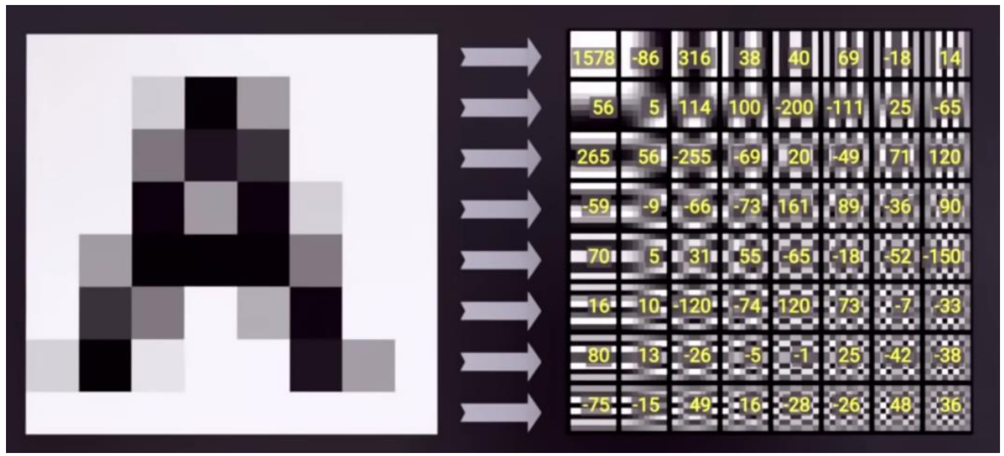
Transform Coding & Quantization

Basis functions



8x8 DCT Transform

Input image | DCT Coefficients

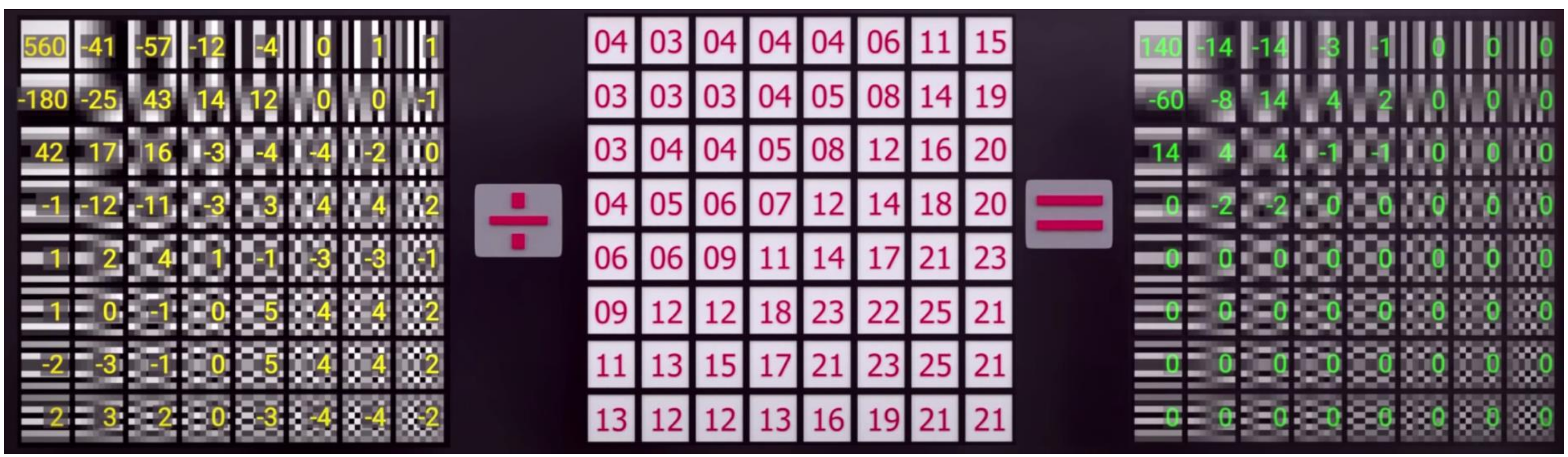


64 constants that represents how much of each base image is used

Transform Coding & Quantization

- Transform encoding and quantization
  - Our eyes are bad at perceiving high frequency data
  - Throw away a lot of such data – negligible quality loss

Lower numbers



DCT image Quantization table Compressed image

Source: vcodexer

Transform Coding & Quantization

04	04	06	10	21	21	21	21
04	05	06	21	21	21	21	21
06	06	12	21	21	21	21	21
10	14	21	21	21	21	21	21
21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21

Chrominance Quantization Table

Higher numbers generate more 0s

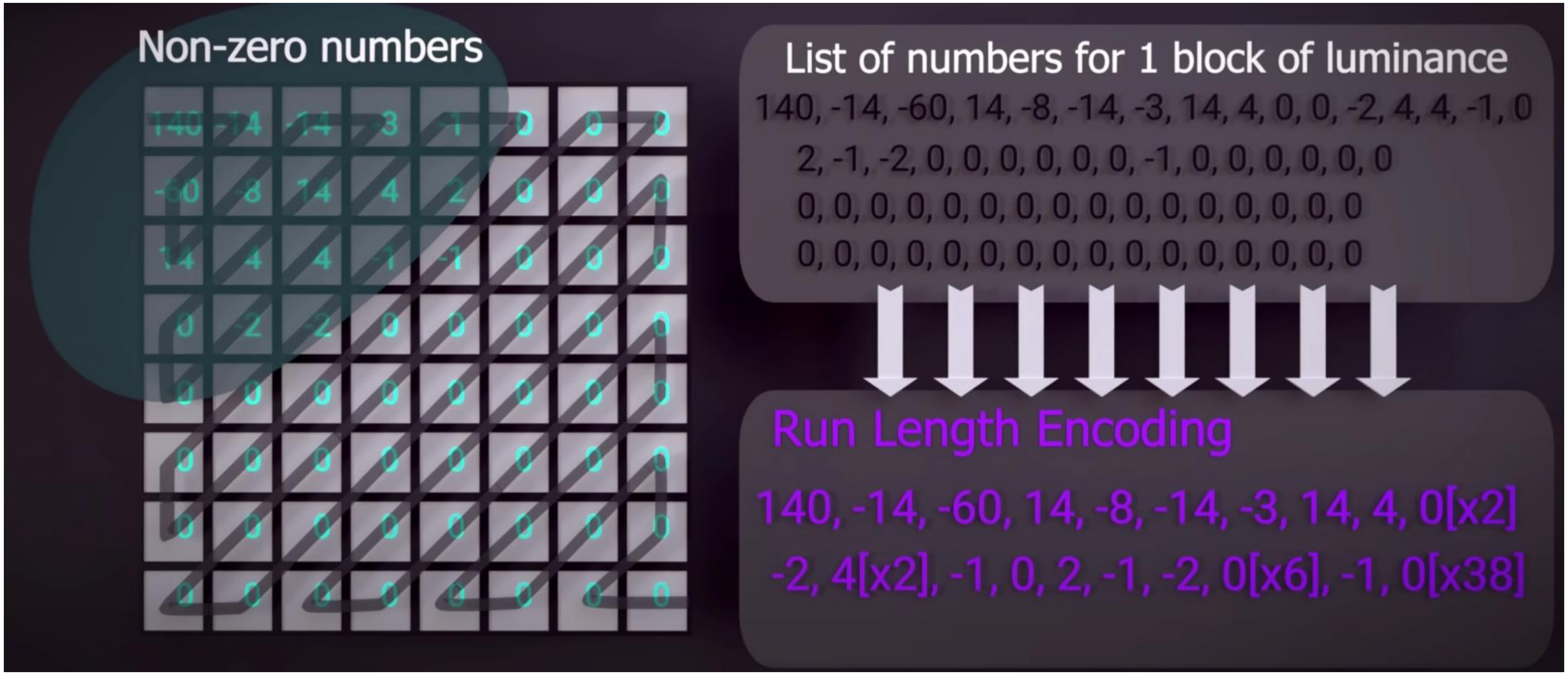
04	03	04	04	04	06	11	15
03	03	03	04	05	08	14	19
03	04	04	05	08	12	16	20
04	05	06	07	12	14	18	20
06	06	09	11	14	17	21	23
09	12	12	18	23	22	25	21
11	13	15	17	21	23	25	21
13	12	12	13	16	19	21	21

Luminance Quantization Table

Lower numbers results in more accuracy

Entropy Coding

- Zigzag Encoding



Entropy Coding

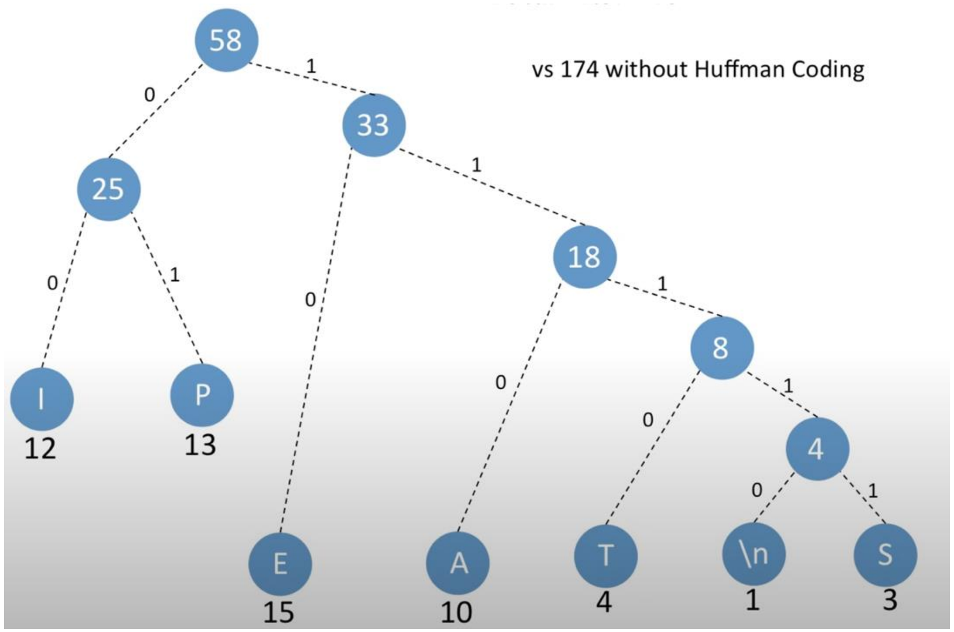
- Huffman coding
  - Based on the lengths of assigned codes on the frequency of data (prefix codes)

Character	Code	Frequency	Total Bits
A	000 Length = 3	10	30 Frequency x Bit Length
E	001	15	45
I	010	12	36
S	011	3	12
T	100	4	12
P	101	13	39
Newline	110	1	3

Total Bits Used: 174

Entropy coding

- Huffman coding



Total Bits: 146

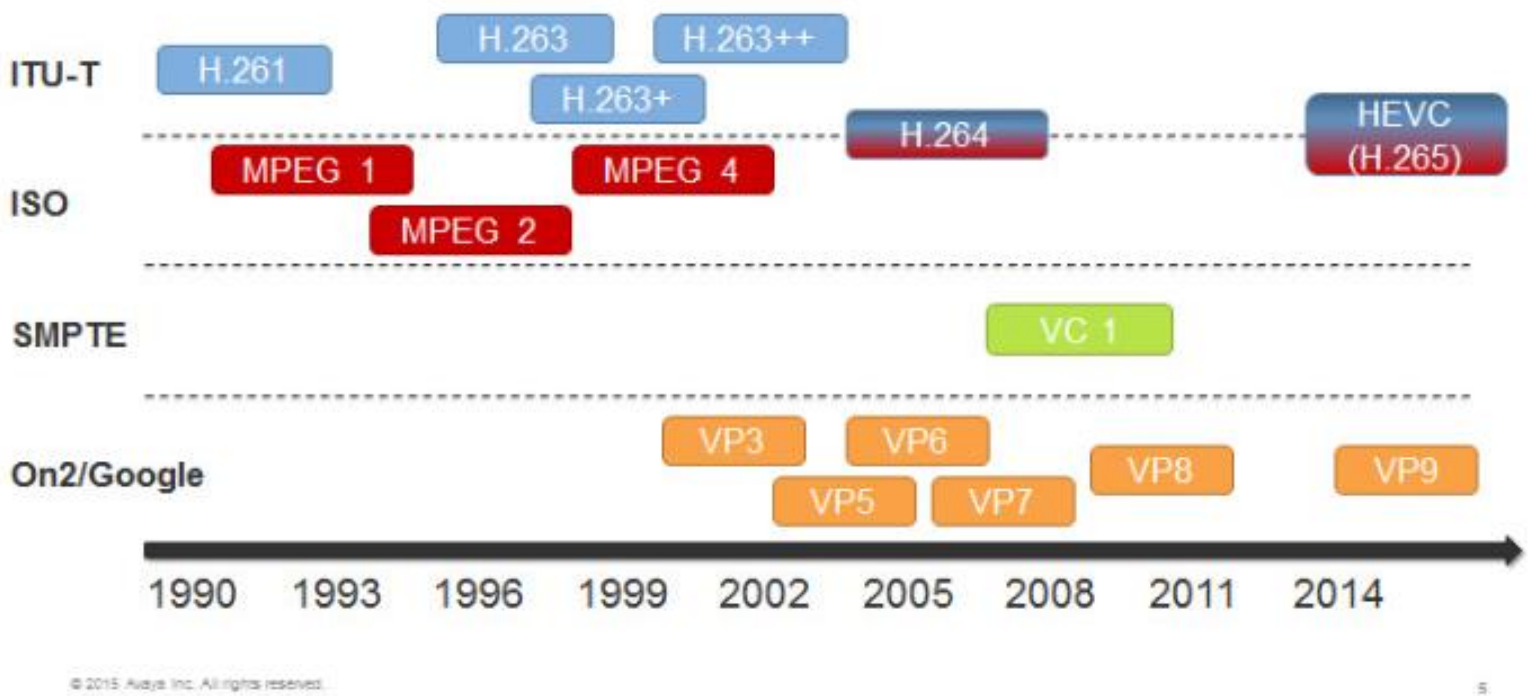
Char	Code	Freq	Total Bits
A	110	10	30
E	10	15	30
I	00	12	24
S	11111	3	15
T	1110	4	16
P	01	13	26
\n	11110	1	5

Compression Artifacts

- 8x8 Blocks

Use deblocking filter

Video Compression History



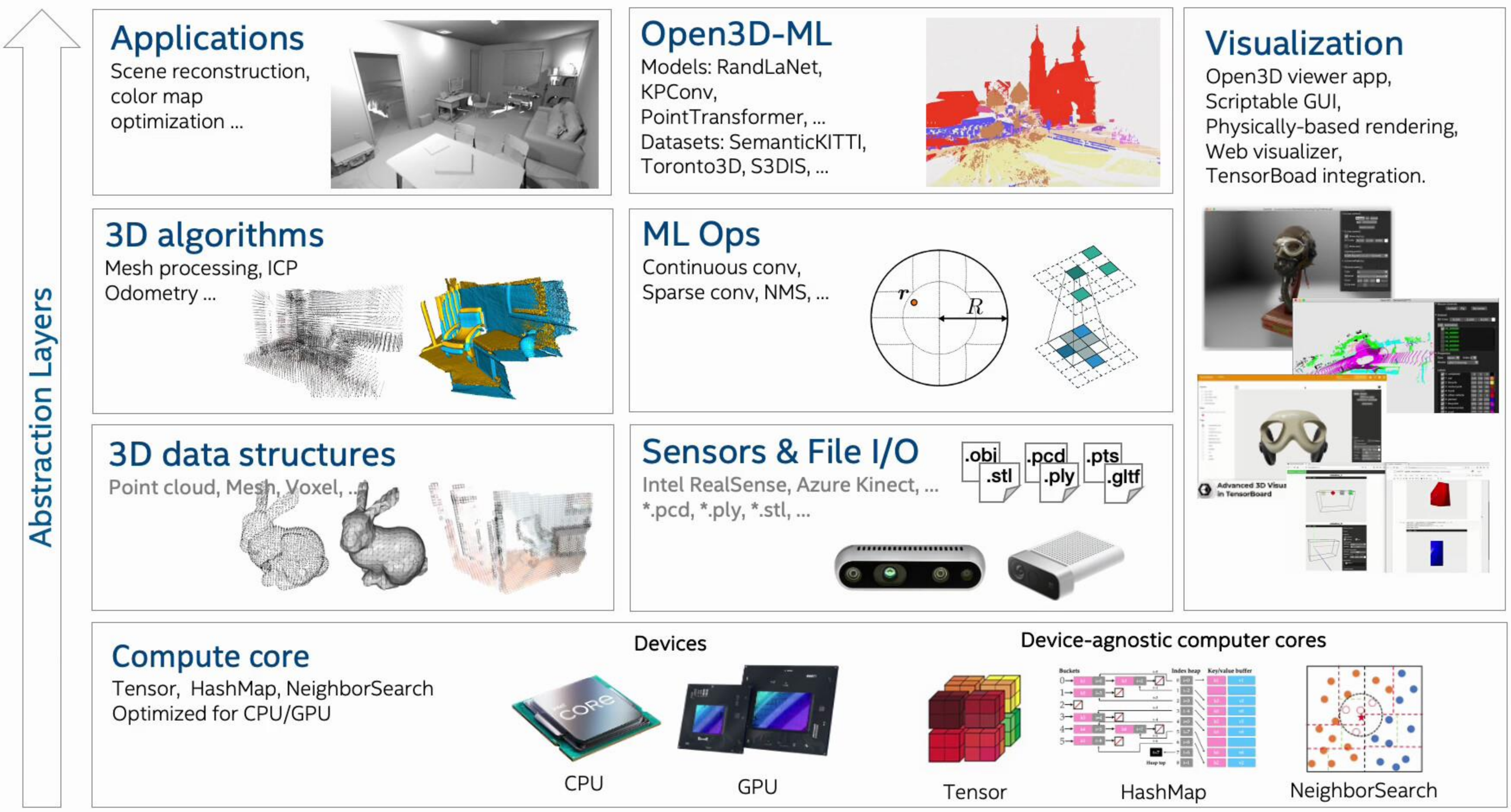
Popular Video Compression Algorithms

- MPEG Standards
  - MPEG H.26x series, H.266 is the most recent one
  - VP series from Google
  - AV1/AV2 this year

Lecture Outline for Today

- 2D Video Compression
- 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, ...
.obj
.stl
.pcd
.ply
.pts
.gltf
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 course 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

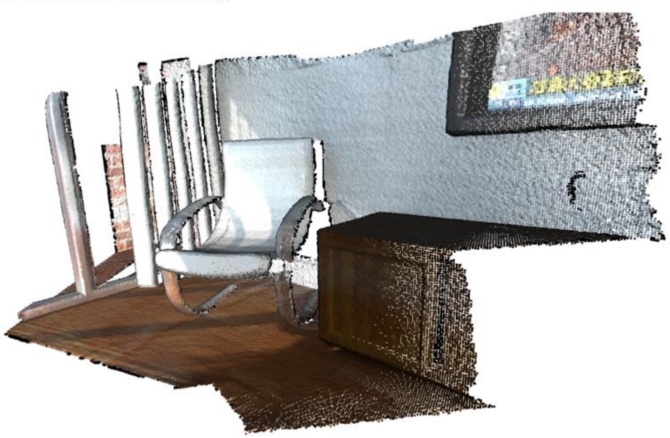
To use Open3D in your C++ project, checkout the following examples

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

Open3D Point Cloud Visualization

```python
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])
```

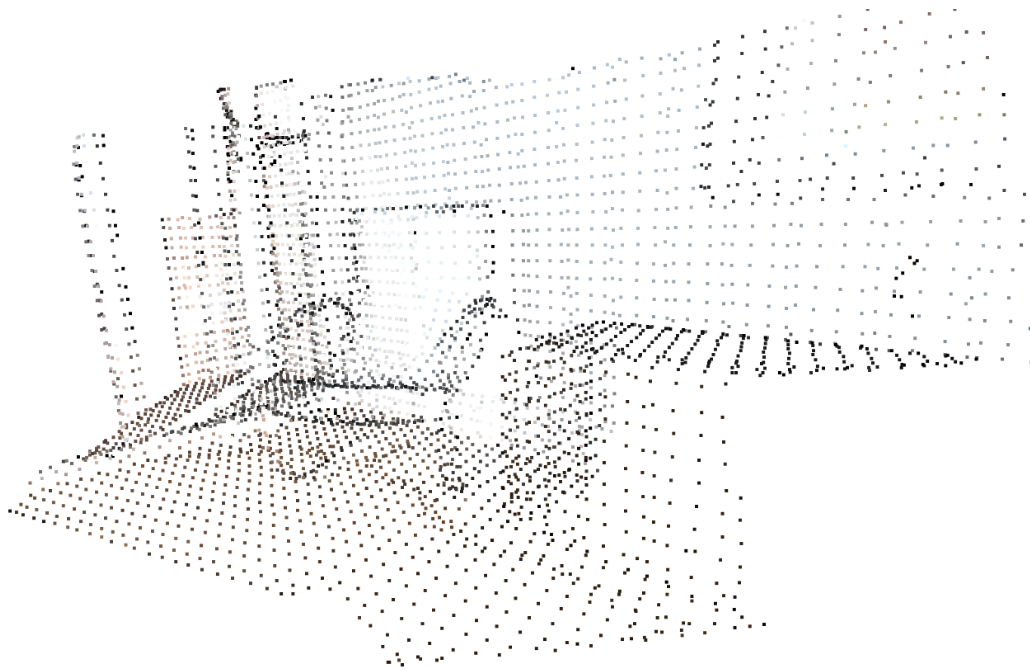
```
Load a ply point cloud, print it, and render it
PointCloud with 196133 points.
[[0.65234375 0.84686458 2.37890625]
 [0.65234375 0.83984375 2.38430572]
 [0.66737998 0.83984375 2.37890625]
 ...
 [2.00839925 2.39453125 1.88671875]
 [2.00390625 2.39488506 1.88671875]
 [2.00390625 2.39453125 1.88793314]]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



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

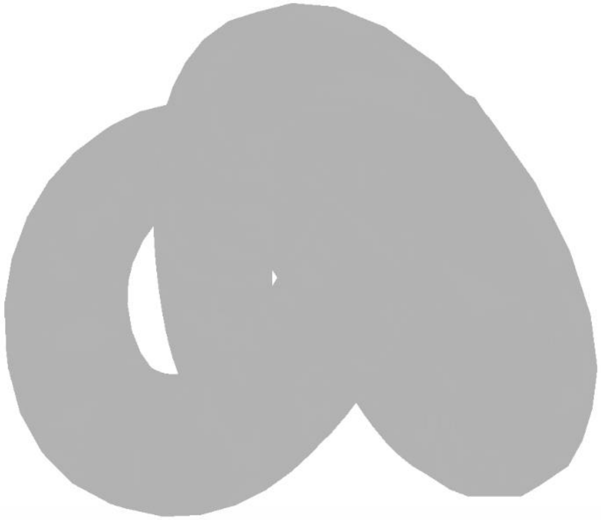
```python
print("Testing mesh in Open3D...")
armadillo_mesh = o3d.data.ArmadilloMesh()
mesh = o3d.io.read_triangle_mesh(armadillo_mesh.path)

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.")
```

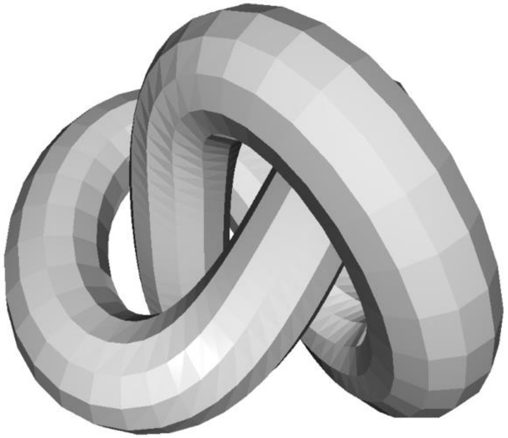
```
Try to render a mesh with normals (exist: False) and colors (exist: False)
```



Open3D Mesh Visualization

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

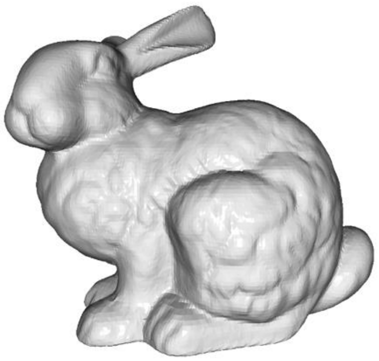
```
Computing normal and rendering it.
[[ 0.79164373 -0.53951444  0.28674793]
 [ 0.8319824  -0.53303008  0.15389681]
 [ 0.83488162 -0.09250101  0.54260136]
 ...
 [ 0.16269924 -0.76215917 -0.6266118 ]
 [ 0.52755226 -0.83707495 -0.14489352]
 [ 0.56778973 -0.76467734 -0.30476777]]
```

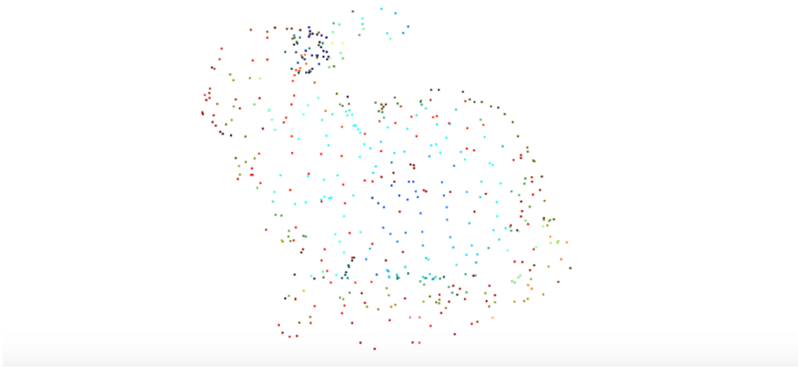


With Normals

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])
```

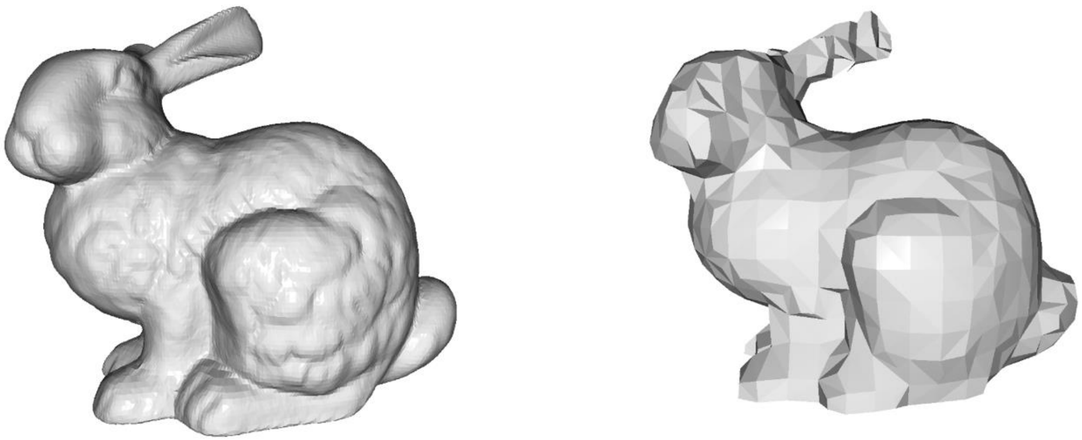




Sampling Point Clouds from Mesh

Open3D Mesh Visualization

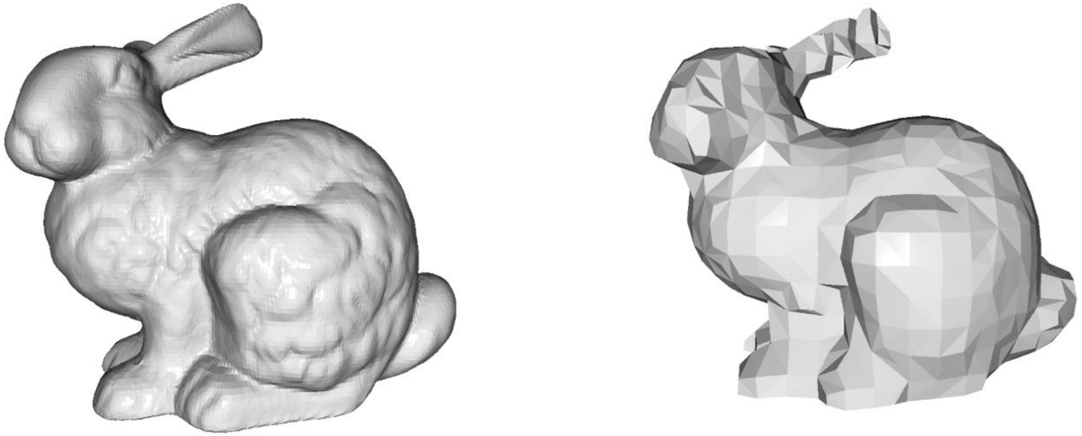
```
voxel_size = max(mesh_in.get_max_bound() - mesh_in.get_min_bound()) / 32
print(f'voxel_size = {voxel_size:e}')
mesh_smp = mesh_in.simplify_vertex_clustering(
    voxel_size=voxel_size,
    contraction=o3d.geometry.SimplificationContraction.Average)
print(
    f'Simplified mesh has {len(mesh_smp.vertices)}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

```python
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

```
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

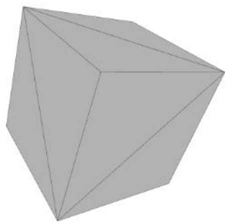
```python
import open3d as o3d
import open3d.visualization.rendering as rendering

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

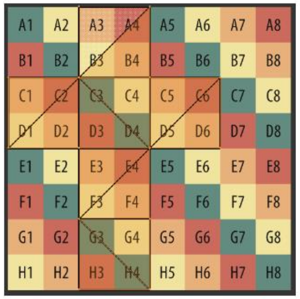
Example Texture Map

Open3D UV Maps

Box (map uv to each face = false)

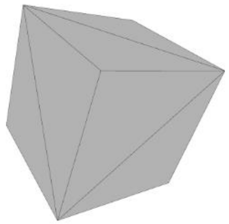
```
box = o3d.geometry.TriangleMesh.create_box(create_uv_map=True)
o3d.visualization.draw({'name': 'box', 'geometry': box, 'material': material})
```

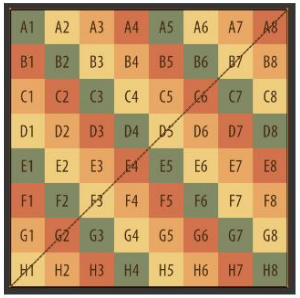


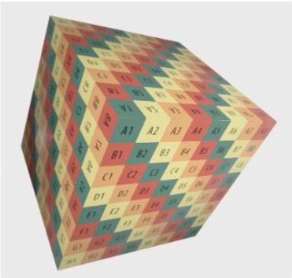


Box (map uv to each face = true)

```
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})
```



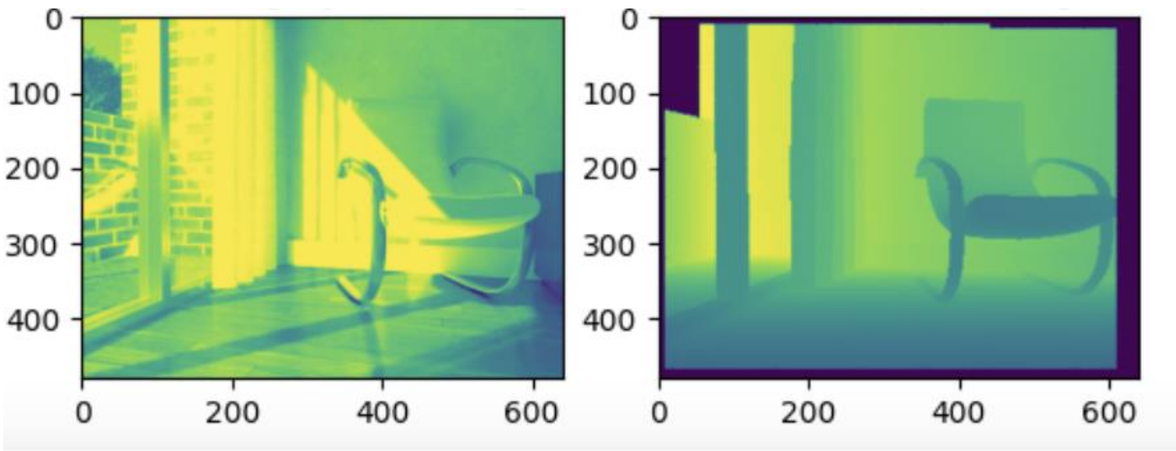




Open3D Depth Map Viewer

```
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()
```



Lecture Outline for Today

- 2D Video Compression
- Open3D
- Depth Map Compression

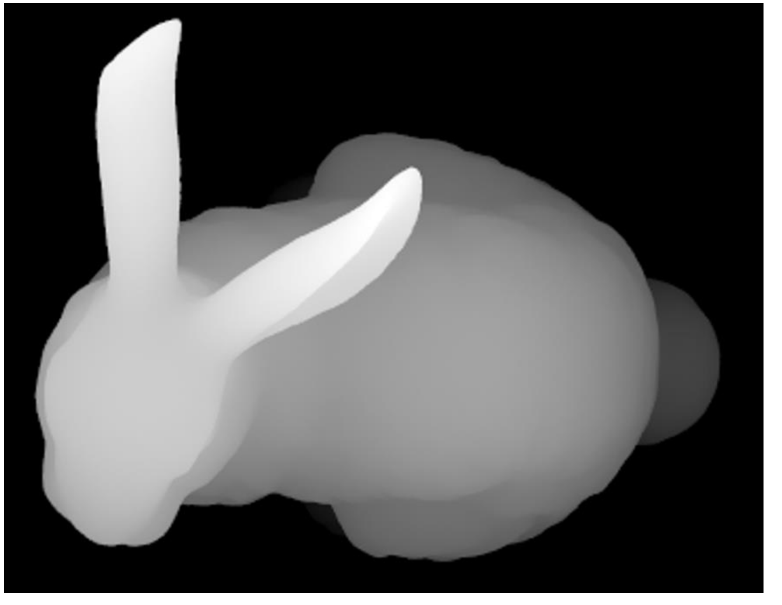
Depth Map Compression

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

- Compute the date rate for a depth video with a resolution of 1024x1024.

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



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.

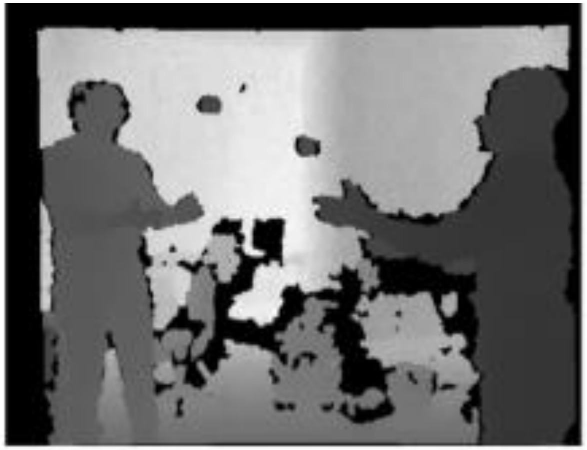
Adopting Standard Video Codecs

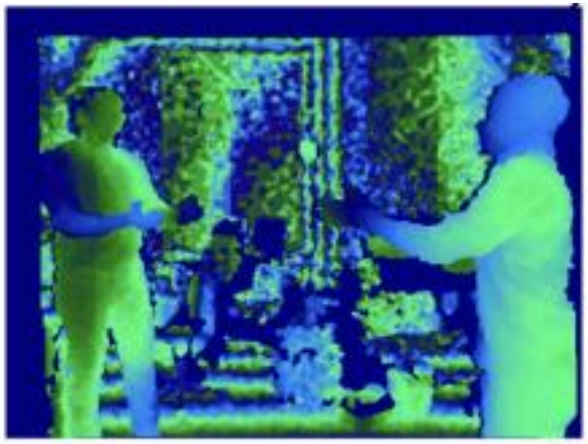
- Challenge
  - Bit-depth and channel inconsistency
  - Depth videos are single channeled
  - Bit-depth of depth videos is larger than color videos in general

  - 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

Bit Packing

- Consider a 16-bit depth value D, which we split across the three 8-bit channels (R, G, B) using the multiplexing method.
- **Original Depth Value**:
  1. Suppose D = 1010101111001101 (binary) or 0xABCD in hexadecimal.
  2. After packing, let's say we have:
     1. **R Channel**: 10101000 (8 bits, representing the first 6 MSBs plus padding zeros)
     2. **G Channel**: 11110000 (8 bits, representing the next 5 bits plus padding zeros)
     3. **B Channel**: 01101000 (8 bits, representing the last 5 bits plus padding zeros)

Bit Packing

- **Quantization During Encoding**:
- When this packed representation is encoded using a lossy video codec, quantization occurs to reduce the data size.
- Imagine that, during quantization, the codec changes a single bit in the **R Channel** from 10101000 to 10100000. This represents a change in one of the most significant bits, dropping the value from 0xA8 to 0xA0.

Bit Packing

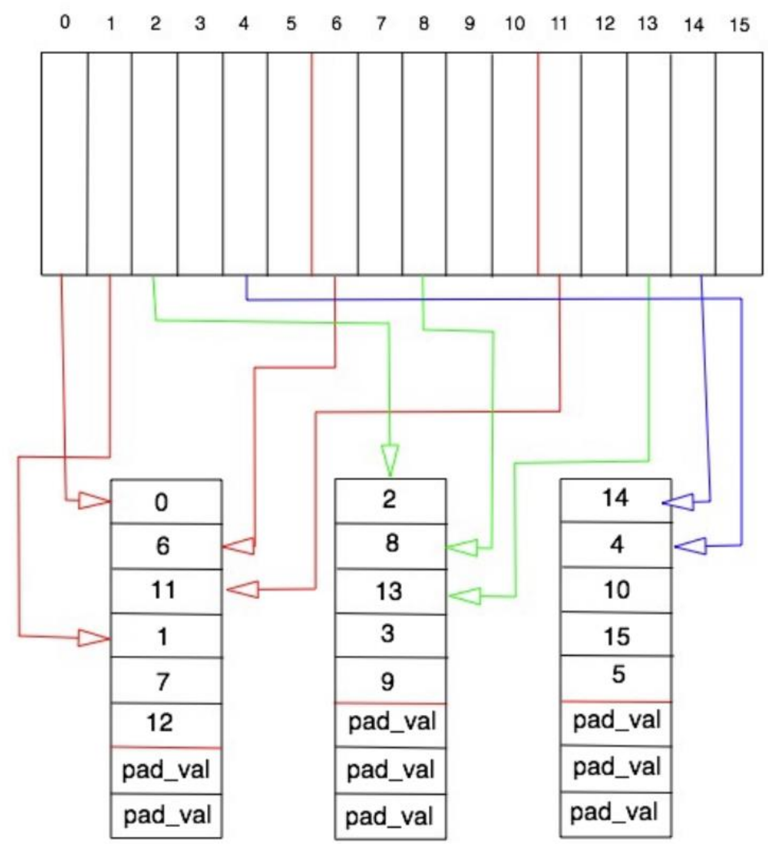
- **Effect on Reconstructed Depth**:
- After decoding, when you try to reconstruct the original 16-bit depth value from the modified R, G, and B channels, you would end up with a different depth value:
- Original **R Channel**: 10101000 (binary) → contributes heavily to the overall depth value.
- Modified **R Channel**: 10100000 (binary) → introduces a significant difference.
- Let's calculate the impact:
    - The original 6 MSBs were 101010, but after quantization, they became 101000.
    - This means the depth value decreased significantly because it is missing 2^5 units in magnitude due to this change.
    - In a depth map context, such a change can represent a substantial shift in the measured distance, potentially several meters if the depth values cover a wide range.

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
  - The first few bits (e.g., 0, 6, 11, etc.) go into one channel.
  - Another set of bits (e.g., 2, 8, 13, etc.) goes into a different channel.
  - A third set of bits (e.g., 14, 4, 10, etc.) goes into the final channel.



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

Adopting Standard Video Codecs

- Interleaved bit packing
  - Since each channel now carries a mix of different bit positions, the impact of compression artifacts is more evenly distributed, potentially making the entire packed value more resistant to the errors introduced by lossy compression.
  - This means that even if some bits are lost or altered, the chance of catastrophic errors affecting the entire depth value is reduced.

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

- 2D Video Compression
- Open3D
- Depth Map Compression

Next up: Point Cloud Compression