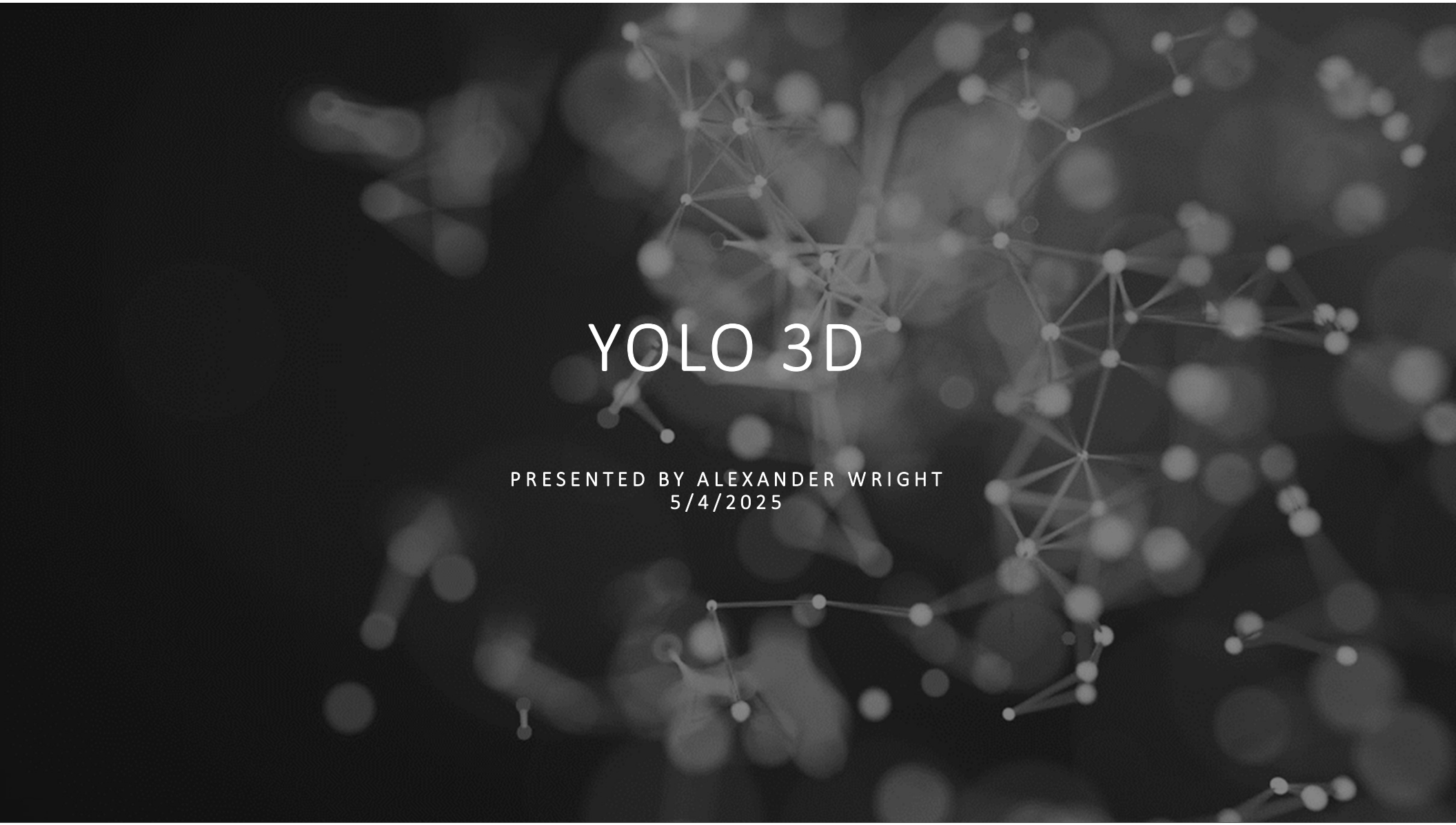


YOLO 3D

PRESENTED BY ALEXANDER WRIGHT
5/4/2025

AGENDA

MOTIVATIONS

YOLO

DEPTH MAPS

NORMAL MAPS

POINT CLOUDS

BOUNDING BOX

PLANE MASKS


MOTIVATIONS

MOTIVATIONS

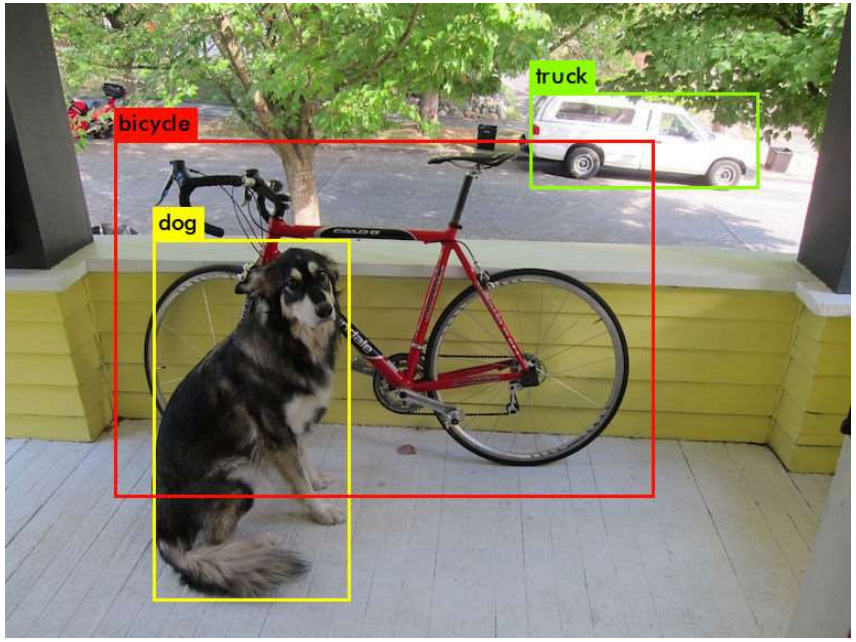
Object Detection in 3D

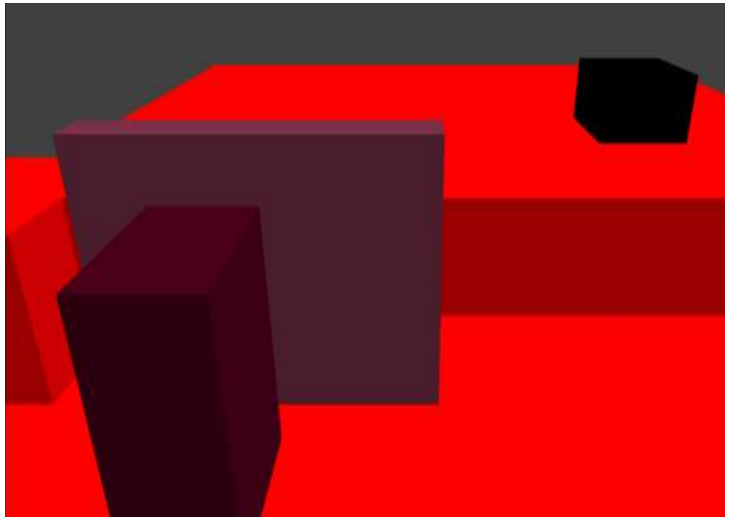
- YOLO is 2D
- Objects in 2D
- For motion planning 3D is desired
  - Self Driving Cars
  - Robotics

Use Cases

- How far away is the car
- How far away is the pedestrian
- Do I need to turn to avoid collision
- How do I position my robot arm in 3d space to pick up this object
- Am I going to collide with anything on my way to this position

YOLO 3D ASPIRATIONS





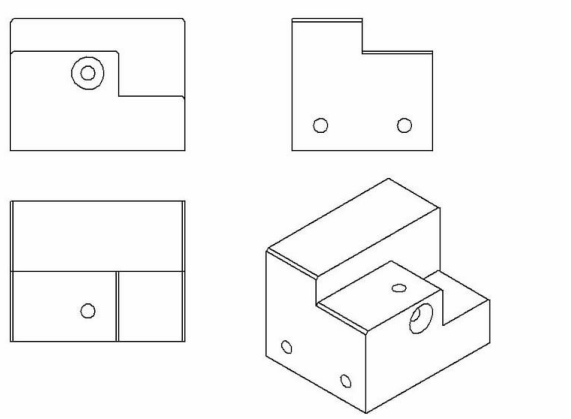
YOLO

YOLO 3D

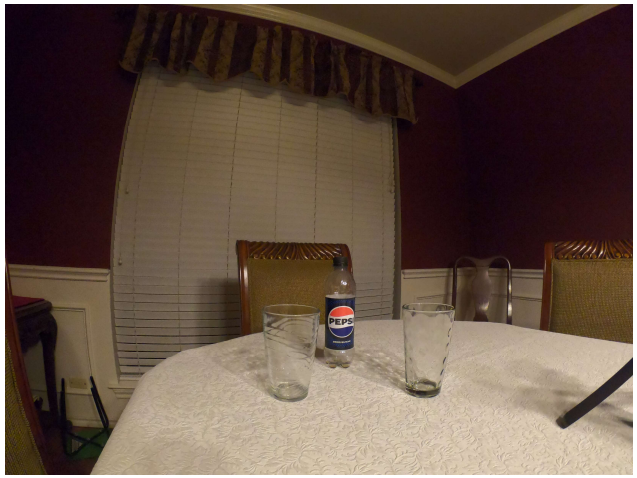
YOLO

YOLO

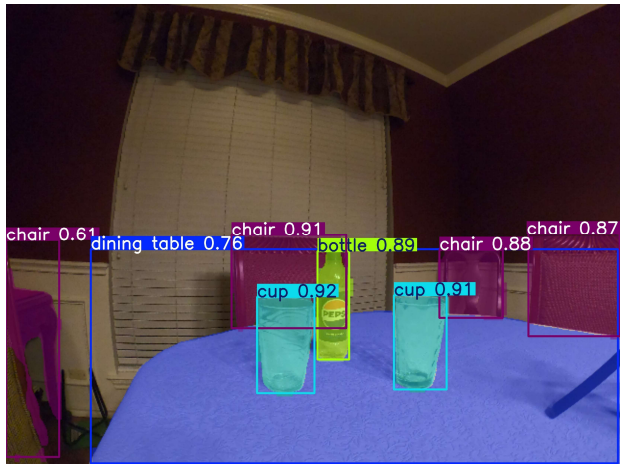
- Detects objects
- Provides bounding box (height, width)
- Can also provide pixel masks
- To get depth we need additional views
  - Side view gives (height, depth)
  - Positioning a camera to the side isn’t practical
  - Settling with two cameras angled inwards

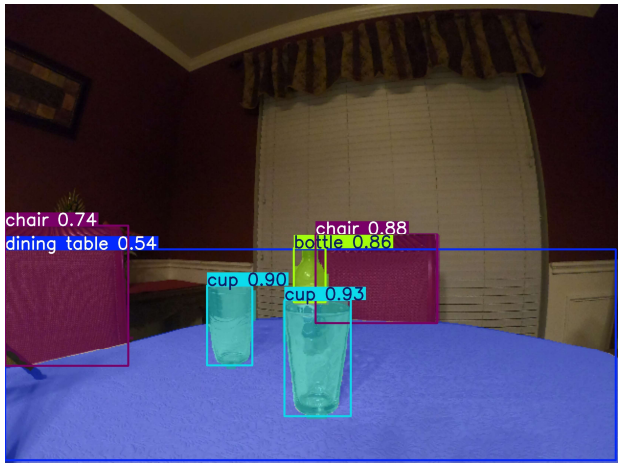


- Two GoPro Hero 7’s on GoPro MaxGrip Stands 12 inches apart
  - 4000x3000 timelapse mode 24 bit RGB
- Angled inward by the max rotation in each direction (twist lock) ~25 to 30 degrees in



- YOLOv11 using Ultralytics API’s



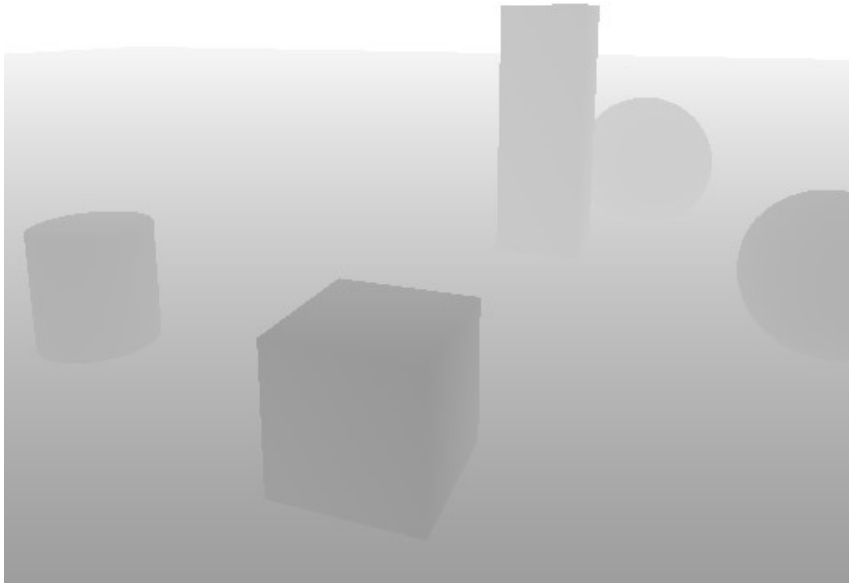


DEPTH MAPS

YOLO 3D

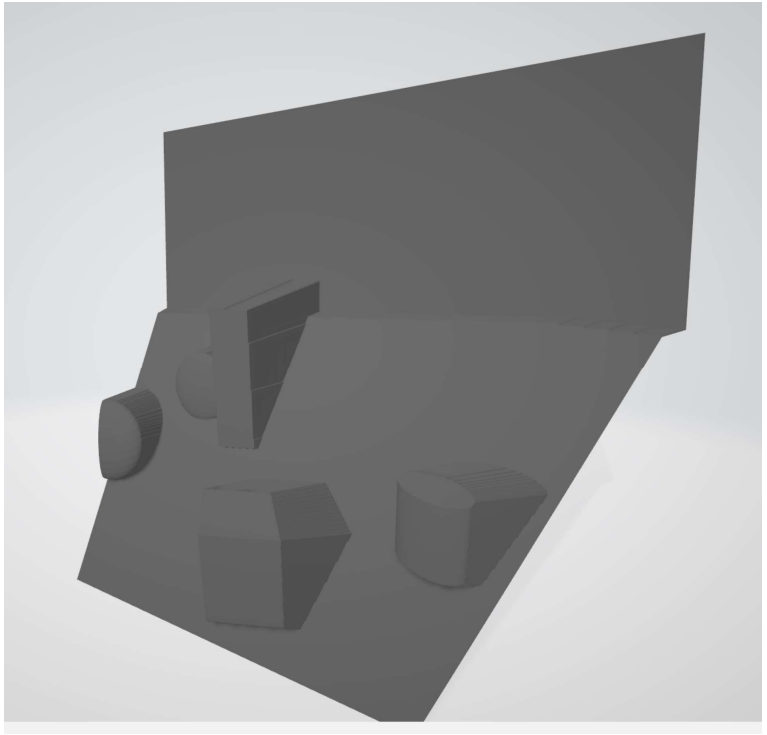
DEPTH MAPS

- Depth Maps
  - Depth Cameras
    - Kinect / Kinect Fusion
    - Intel Real Sense
    - Stereolabs Zed2
  - Computer Graphics
    - Z-buffer
- Distance from the viewer



DEPTH MAPS AS GEOMETRY

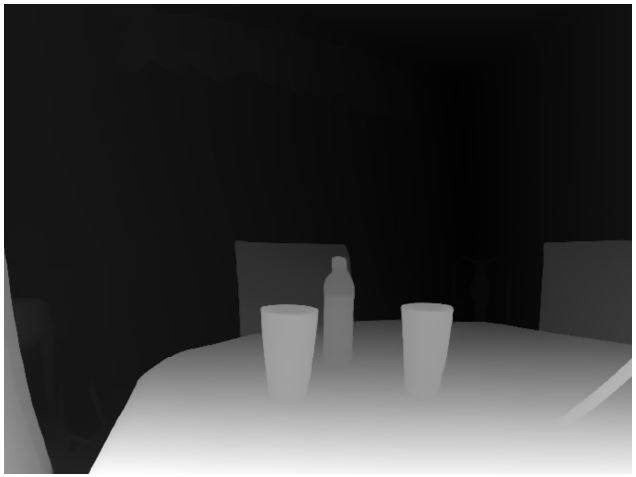
- Can be converted into a mesh
  - (X,Y) pixel position
  - Z Depth into image
  - Connect all points into triangles
  - Can also leave unconnected as a point cloud
- From triangles, you can generate normal maps
  - Cross Product of triangle edge vectors

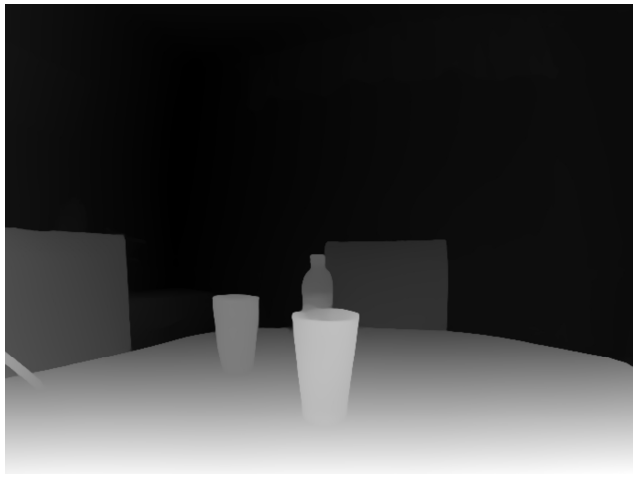


DEPTH ANYTHING V2

- Takes RGB image and generates a depth map
  - Models use Image CNN's and match against Depth Datasets
  - They assume it works by measuring the objects pixel heights against known heights
  - Two modes exist, normal and metric
  - Metric is far more accurate, but has different training for indoor vs outdoor
- Now we may no longer need an expensive depth camera

- Depth Anything v2
  - Note horizontal lines are some issue with PDF generation from slides.
  - (Don’t really exist)





NORMAL MAPS

YOLO 3D

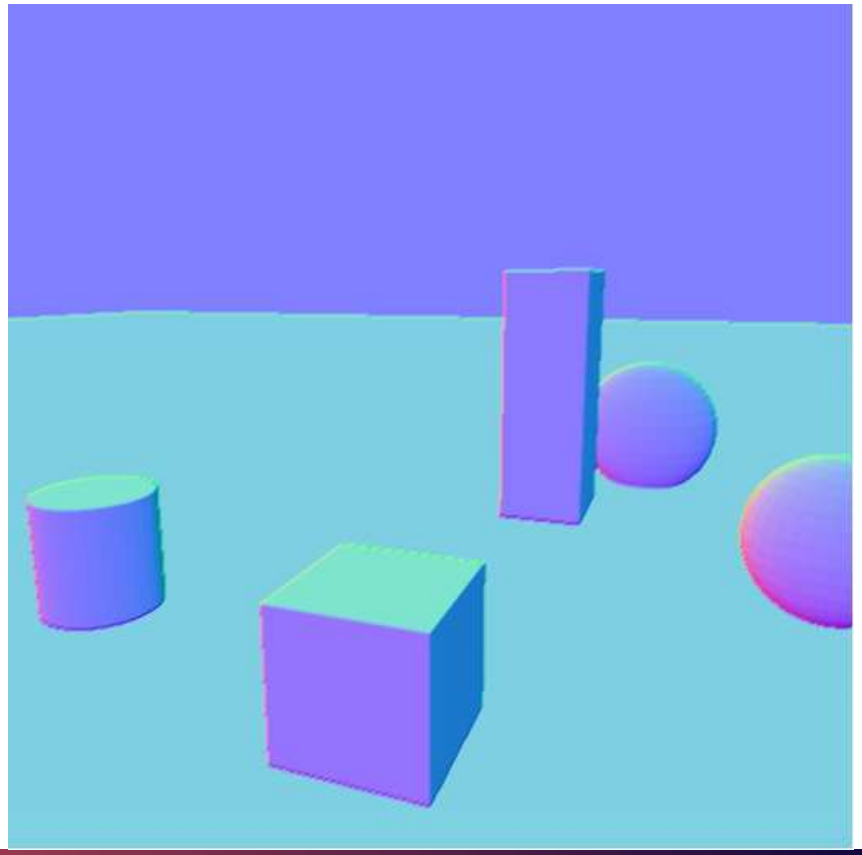
NORMAL MAPS

- Normal Maps
  - Represents surface orientation
  - Used in lighting in computer graphics

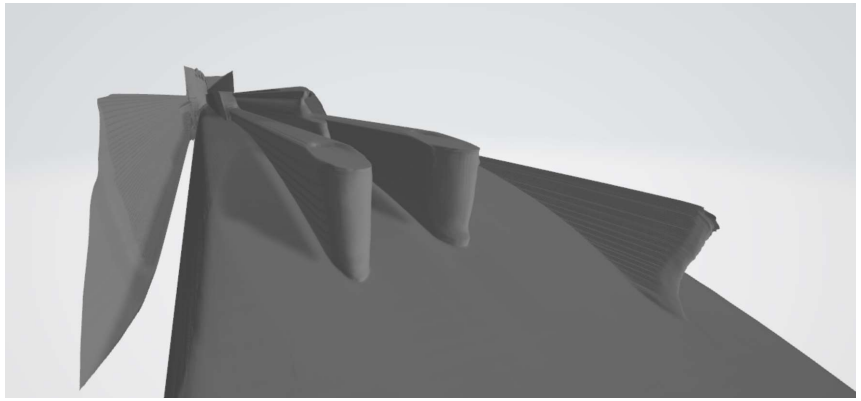
  Plane equations

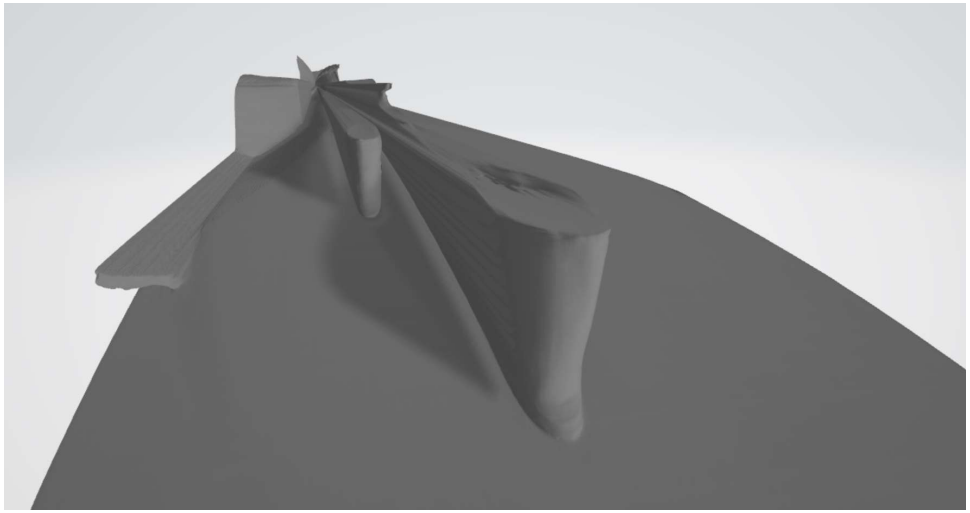
  Need A,B,C,D that satisfy
  - Ax + By + Cz + D = 0
  - (A,B,C) is the normal
  - D is the normal dot a point on the plane
  - Point Normal Form:
    - A(x-x0)+B(y-y0)+C(z-z0) = 0

  (x,y,z) are just free variables, when they satisfy the equation we are on the plane

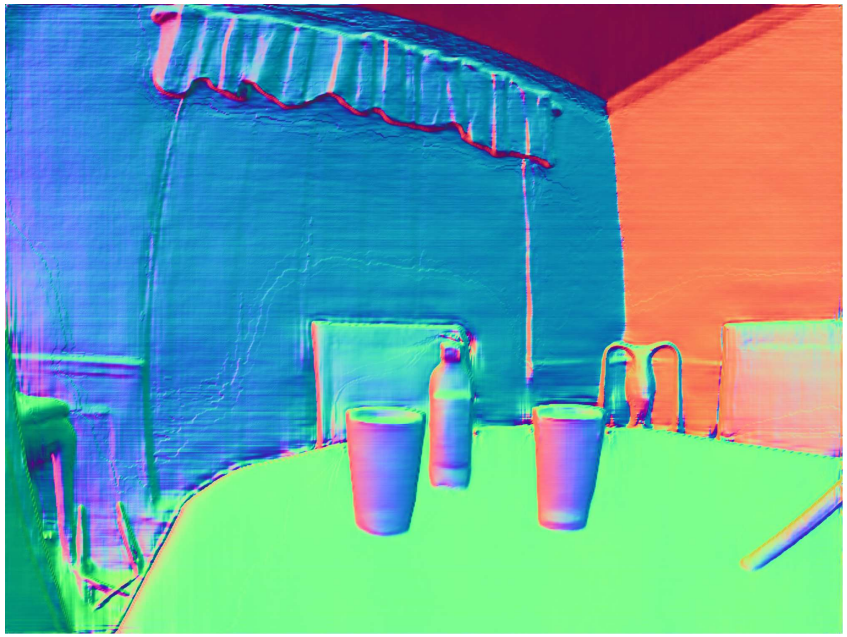


- Mesh From Depth Map
  - Scaling per depth value
  - Total Depth
  - Height isn’t really Y pixel position (table doesn’t go up in space)
  - Z = 1 / Depth versus Z = Depth directly





- Convert to triangle mesh, generate normal maps
  - Note: reduced size from 4000x3000 to 2000x1500 due to python crashing
- Very slow
  - Might need to further reduce image size or optimize python code (broadcasting loops)



POINT CLOUDS

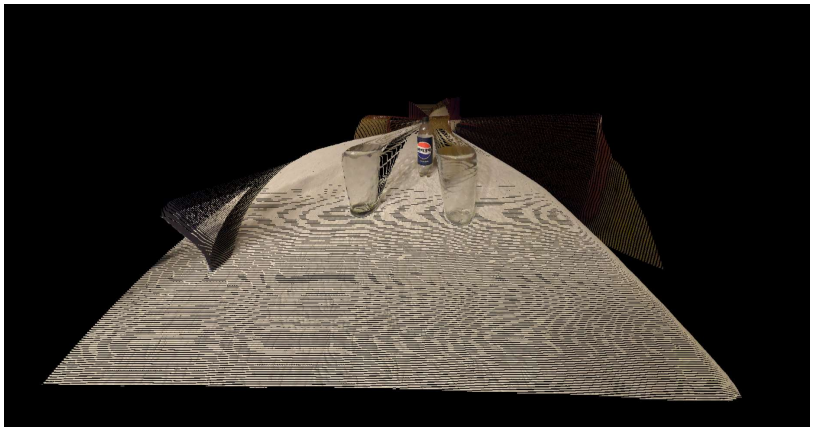
YOLO 3D

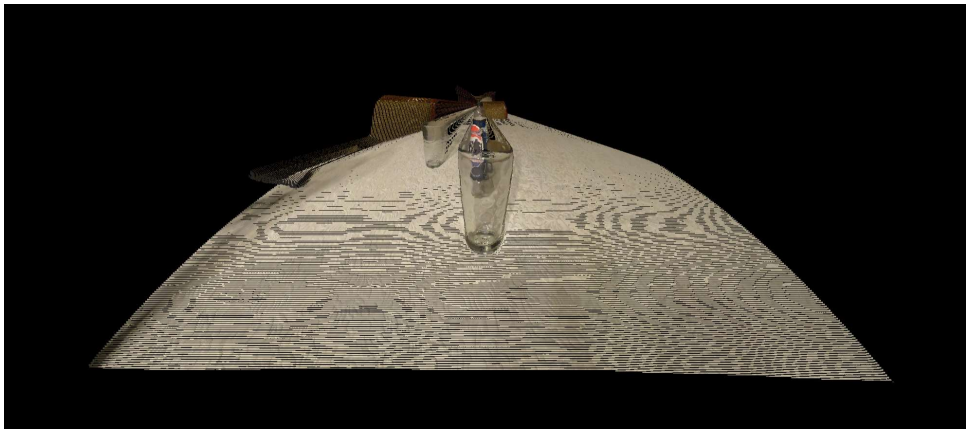
POINT CLOUDS

- Think of each pixel as a point / vector of data about that point
  - (X,Y,Z) 3d point
  - (X,Y,Z,R,G,B) 3d point with color
  - (X,Y) pixel position in the image
  - Z – Depth from depth anything
  - RGB – Color from original image
  - Normal (Nx,Ny,Nz) – Can be used to determine floors, walls, ceilings
  - Object Class (integer) – Object type for pixel from YOLO (masking version)
  - To get floors, walls, ceilings, we need to convert normal to class integer mask

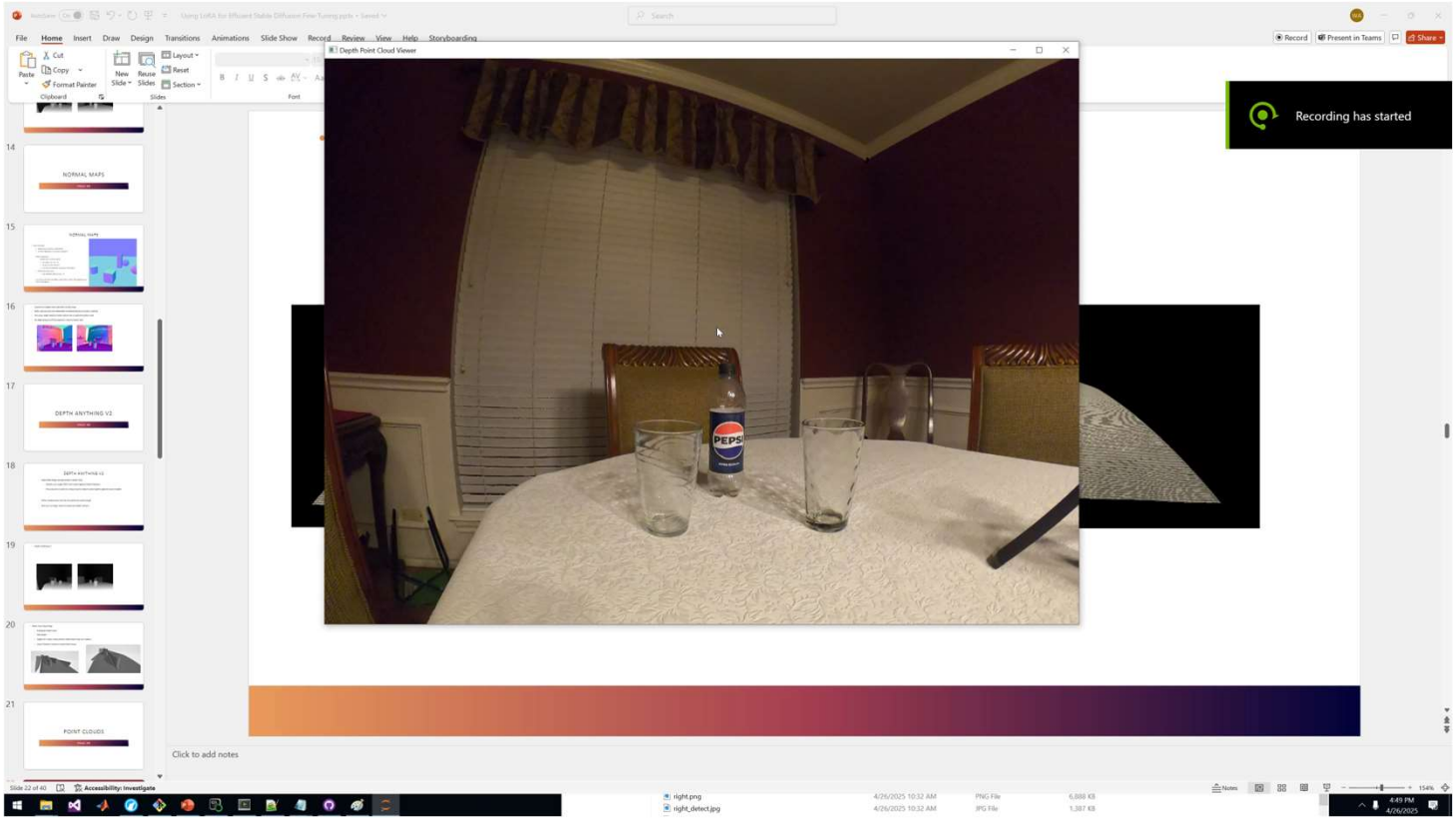
```
(X, Y, Z, R, G, B, Nx, Ny, Nz, class)
```

- Point Cloud visualization MATLAB
- Wasn’t too happy with the visualization
  - Normals generated were in frame, should be able to get point cloud in frame too
  - Was using Depth directly here, should be 1 / Depth (less elongated) for non-metric DepthAnything

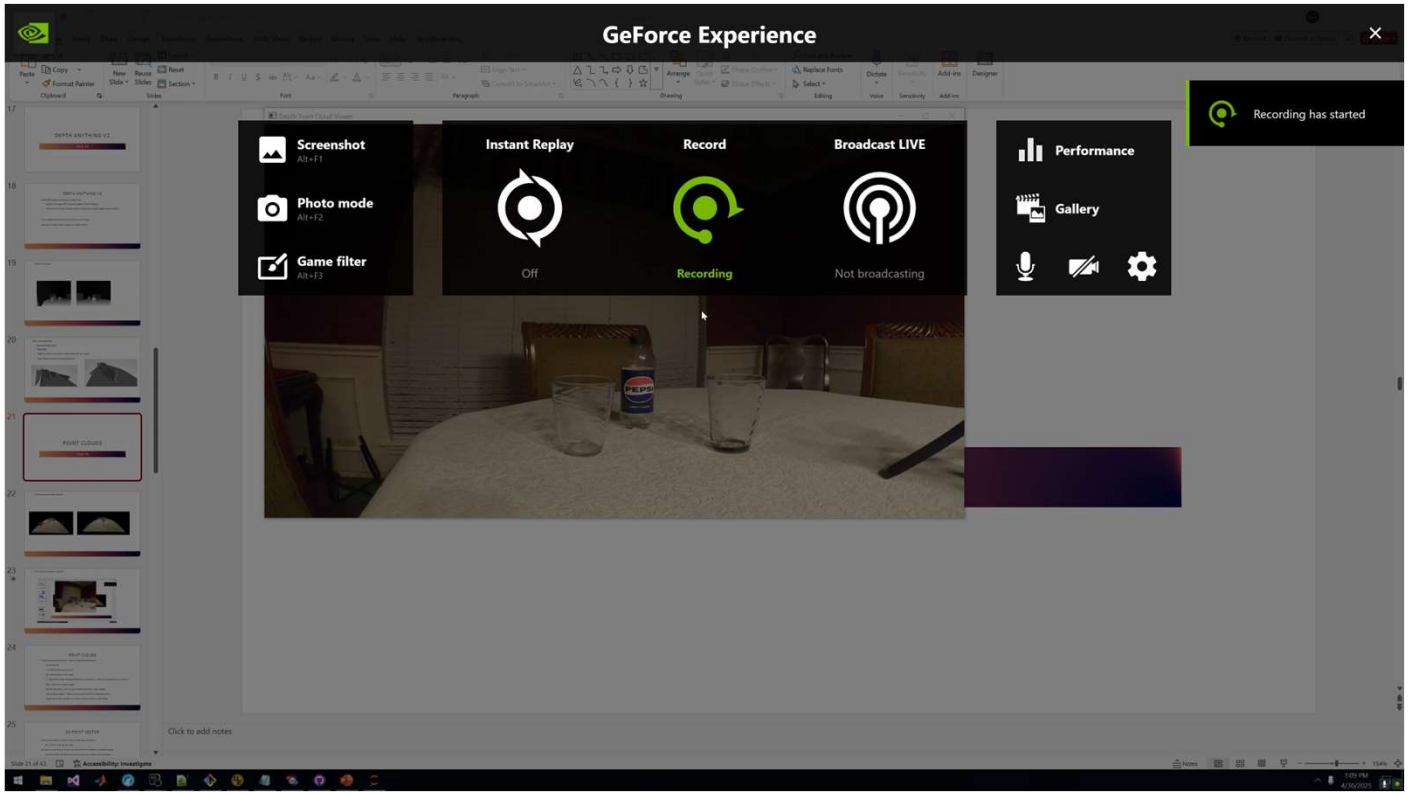




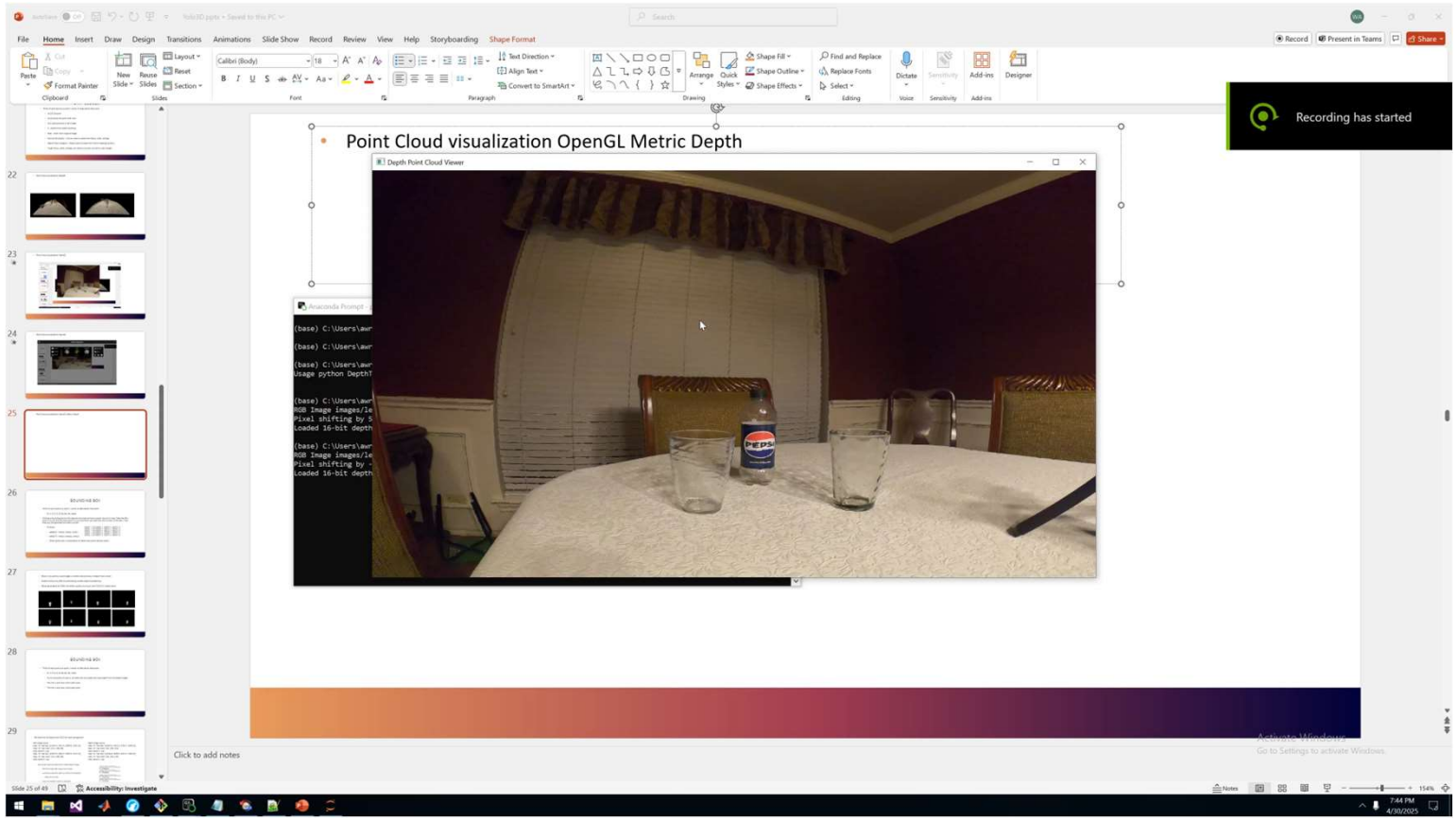
- Video: Point Cloud visualization using Python and OpenGL
  - Still using depth directly ie: Z = Depth



- Video: Point Cloud visualization using Python and OpenGL
  - Z = 1 / Depth (non-metric version of depths)
  - Room is still elongated (chair in corner and chair against window don’t have lots of space behind them)



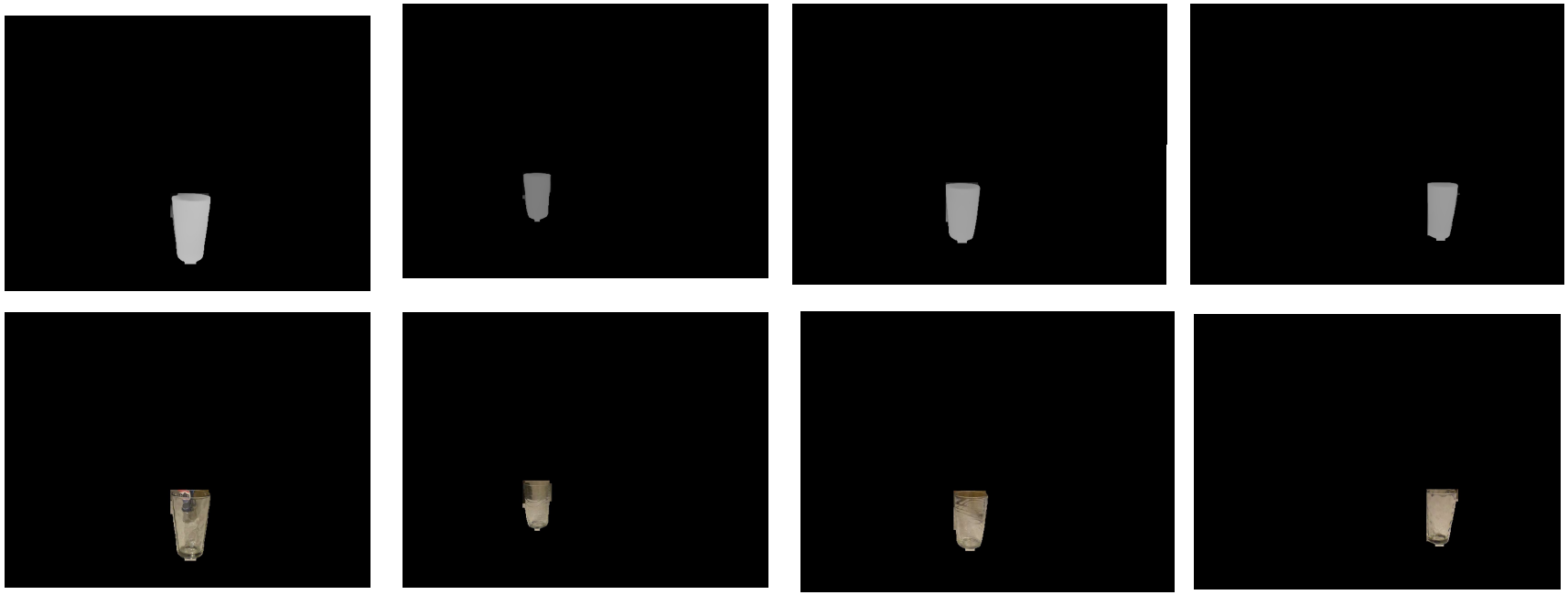
- Video: Point Cloud visualization OpenGL Metric Depth
  - Z = Depth (for metric) Note space behind chair and window more correct



OBJECT MASKS

- Mask is not perfect, avoid edges or detect discontinuity in depth from center
- Scaled contours by 80% to avoid being outside object (isolate3.py)
- Masks generated by SegmentAnything2 (SAM2) are better quality, but stuck with YOLOv11 scaled down

OBJECT MASKS



BOUNDING BOX

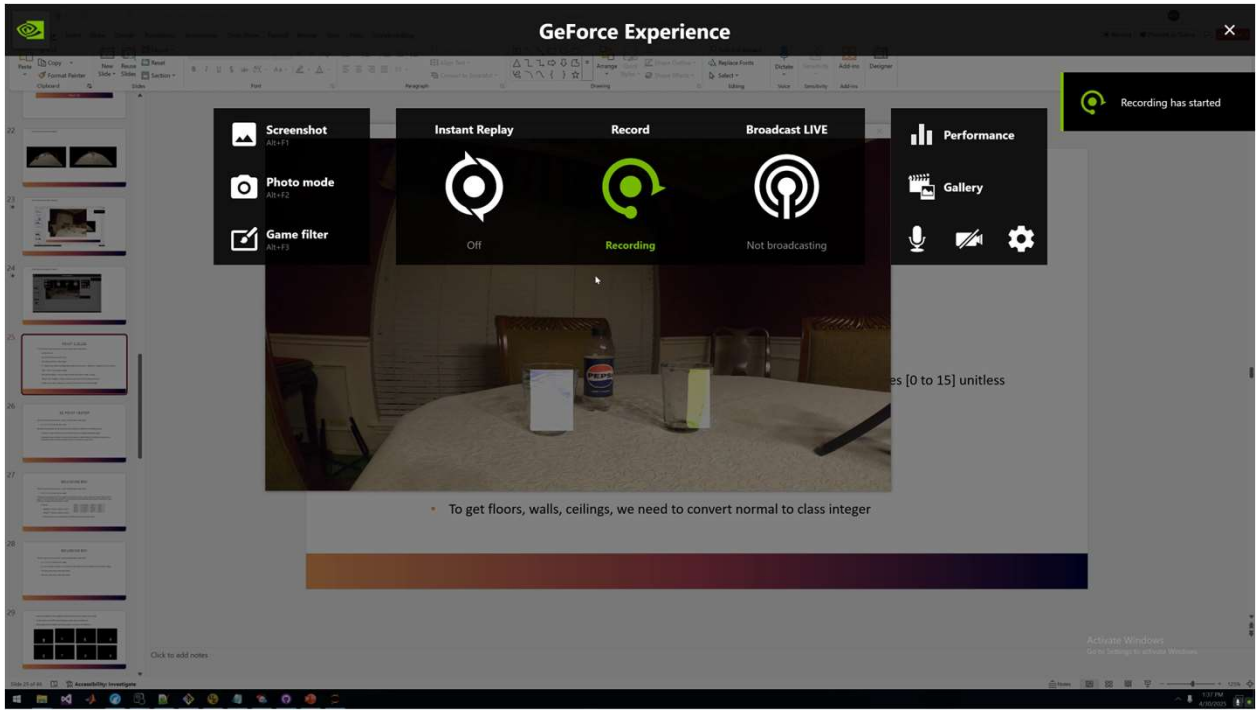
- Getting a bounding box for the objects now that we have a point cloud isn't bad. Take the Min point in (x,y,z) and the max point in (x,y,z) and then you have the two corners of the box. From that you can generate the other corners
  - 8 Points:
    - aabb[0] = min(x), min(y), min(z)
    - aabb[7] = max(x), max(y), max(z)
    - Other points are a combination of these two points (binary order)

```
aabb[1] = vec3(aabb[0].x, aabb[0].y, aabb[7].z);
aabb[2] = vec3(aabb[0].x, aabb[7].y, aabb[0].z);
aabb[3] = vec3(aabb[0].x, aabb[7].y, aabb[7].z);
aabb[4] = vec3(aabb[7].x, aabb[0].y, aabb[0].z);
aabb[5] = vec3(aabb[7].x, aabb[0].y, aabb[7].z);
aabb[6] = vec3(aabb[7].x, aabb[7].y, aabb[0].z);
```

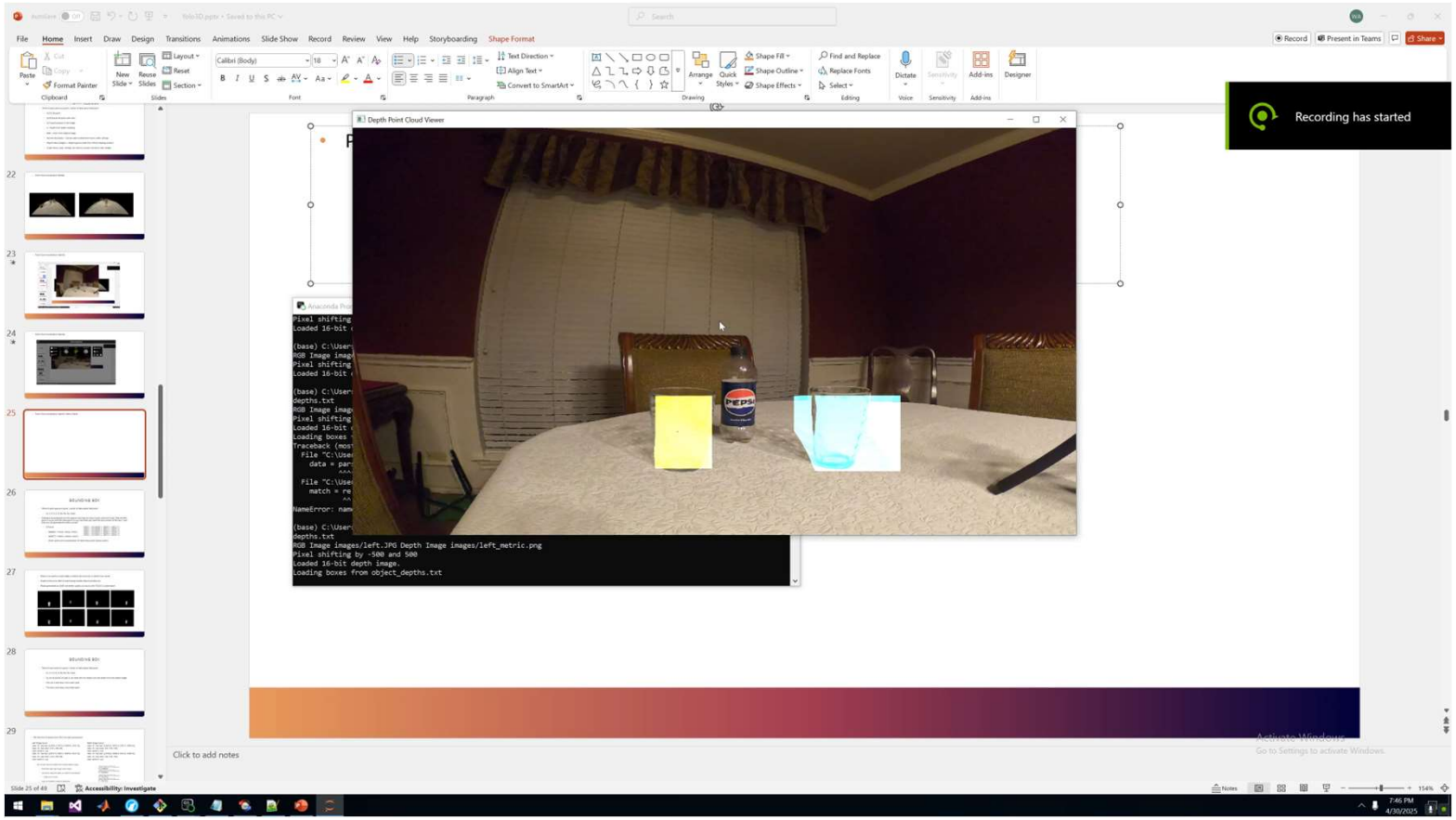
BOUNDING BOX

- Bounding Box
  - So, for all points of class X, we need the min depth and max depth from the depth mask.
  - The min x and max x from each pixel
  - The min y and max y from each pixel
  - Ignored 2D box generated by YOLO

- Video: Point Cloud visualization OpenGL with AABB – (Non-Metric)



- Video: Point Cloud visualization OpenGL with AABB (Metric)

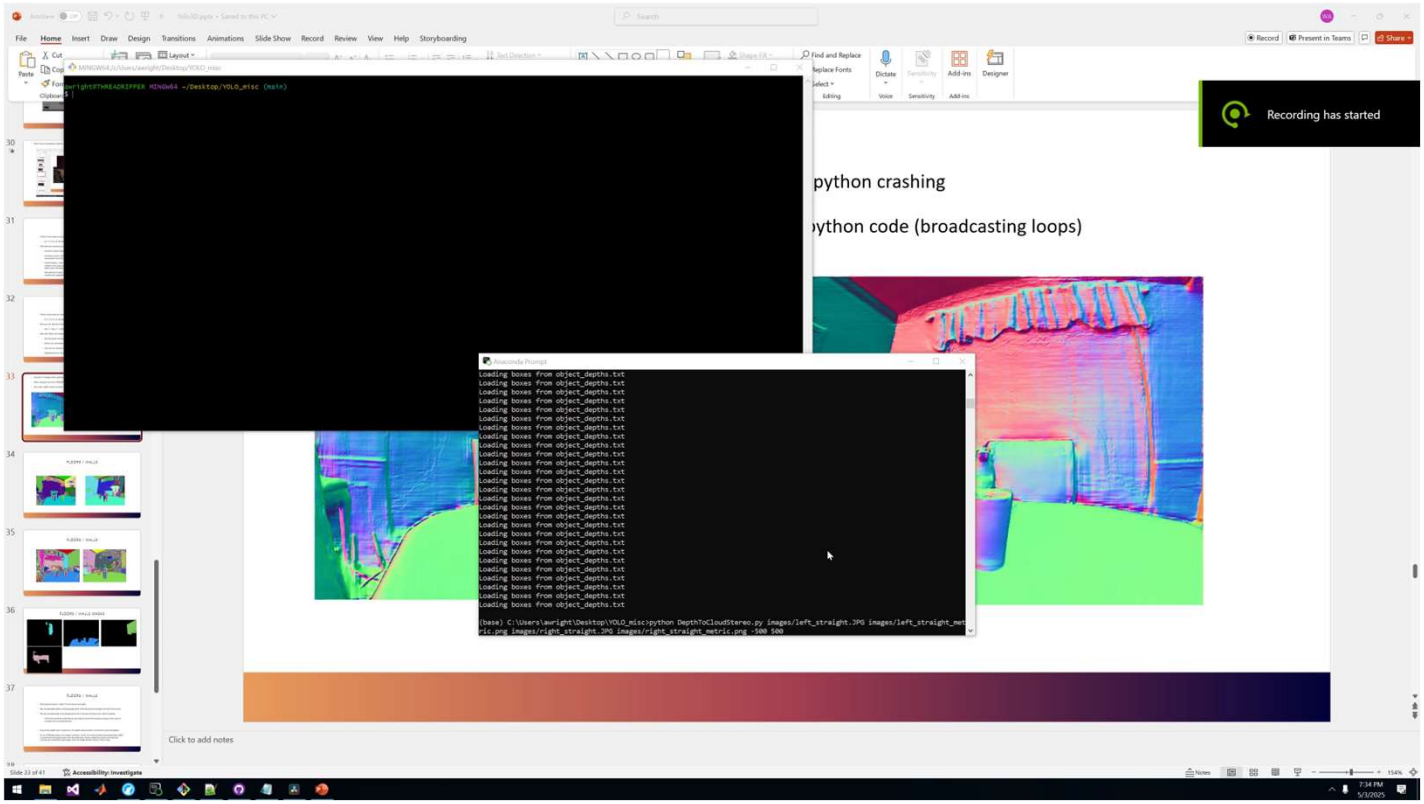


STEREO 3D POINTS

- We have two cameras so we have two sets of points in different coordinate spaces
  - Constant Linear transform to move both sets into a single coordinate space
  - If camera moves, SLAM / Structure from Motion, Object Pose estimation can extract the movement from the point clouds. (ie: COLMAP) But for our case we wont move.
  - Unfortunately, I was not able to get the XY axis (pixels) to correspond nicely with 30° rotation inwards due to the depth having distortion (elongated).
  - Metric depth was noticeably better from this perspective, but I switched to horizontal only stereo images.

- Point Cloud Stereo visualization OpenGL Metric Depth AABB
  - Changed setup from 30° rotation inwards on two tripods to single tripod without rotation
  - Due to depth elongation preventing good overlap (pixels versus depth value units differ enough)
  - Single tripod with horizontal offset only (fixed offset is constant)

- Video: Point Cloud Stereo visualization OpenGL Metric Depth AABB
  - Initial shifting is constant and can be stored in code

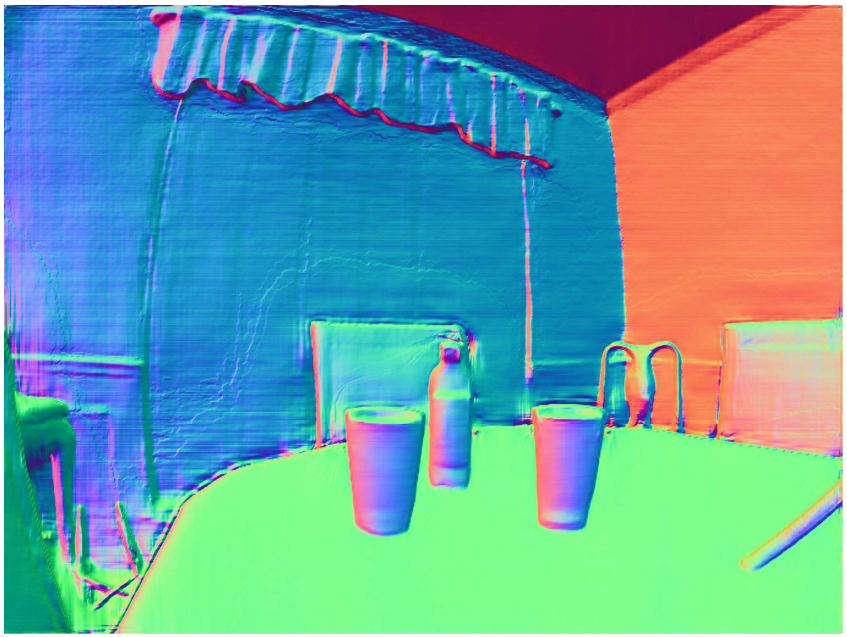


FLOORS / WALLS

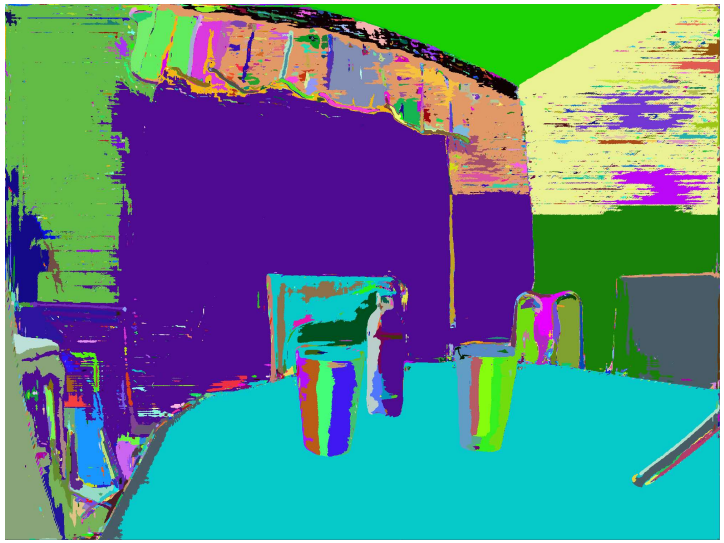
- We can take the normal (each 32 bit floats) and convert them into a class
  - Say 1 = floor, 2 = ceiling, 3 = Wall facing positive X, etc
- We want these two properties
  - Normal points the same direction to some threshold deviation
  - Points are connected (we can have a balcony floor looking over a ground floor)
  - Can sort by number of pixels so we have the most prominent flat surfaces
  - Keeping top four groups by pixel count

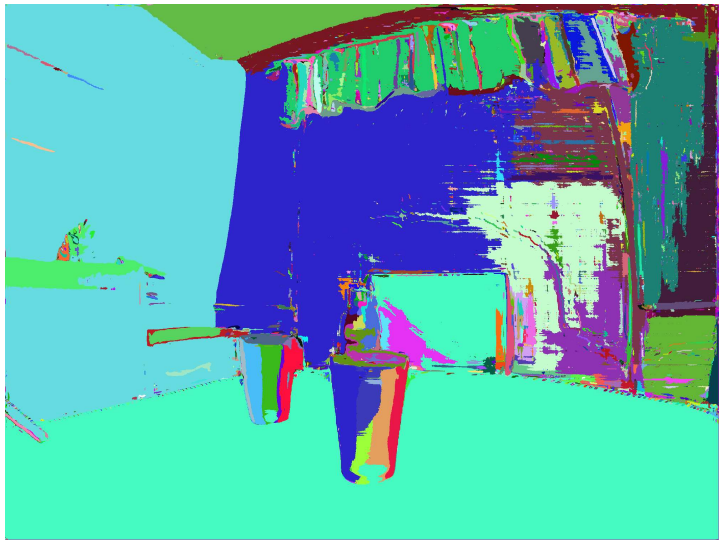
NORMAL MAP FROM BEFORE

- Non-Metric Normals (below) seemed smoother
- Metric Normals seemed to have more noise

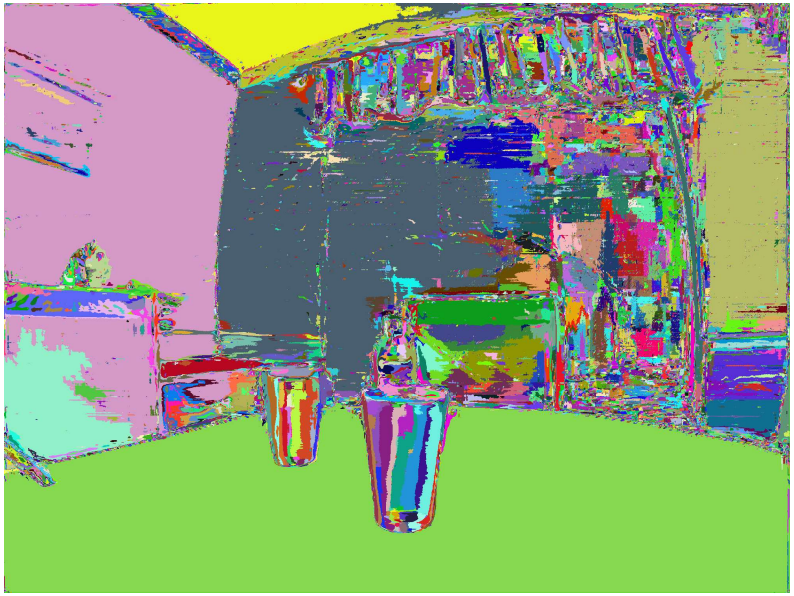


GROUP IN 10 DEGREE COMMON VECTORS





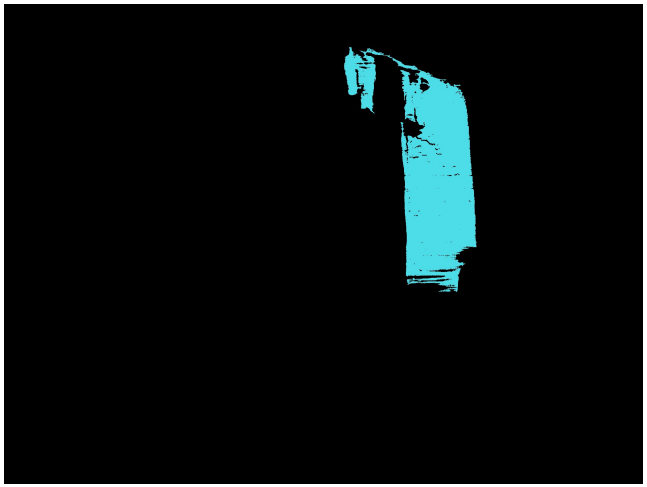
GROUP IN 5 DEGREE COMMON VECTORS

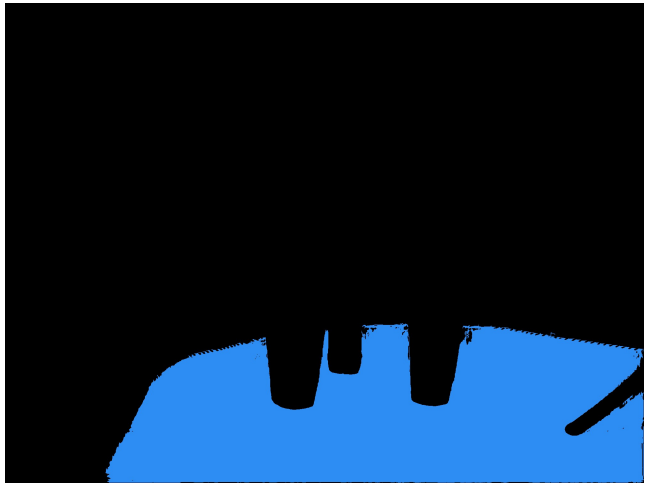


FLOORS / WALLS MASKS

- Now that we have groups of common directions
  - Keep the largest groups that should correspond with large planes

FLOORS / WALLS MASKS





FLOORS / WALLS

- Now that we have a “mask” for the floors and walls
- We can generate planes using average point of the group and average normal of the group
- We can also generate a bounding box by the min/max technique we used for objects
    - Think this would be preferred as we’ll have to bound the planes anyway in the case of multiple floors being detected
- Due to the depth lack of precision, the depth values further out become quite elongated
- So, an AABB gets quite a bit elongated as well, however, I think if we had oriented bounding boxes (OBB) it would look fairly good even with the distortion. Simply rotate the points such that the normal and coordinate space align, then the AABB should conform more nicely

FURTHER RESEARCH IDEAS

- Generate our new vector [R,G,B,D,Nx,Ny,Nz,class] on ImageNet to feed a 2D/3D CNN and see how it performs
  - Maybe just run on basic Cat/Dog to see if it can provide an improvement using MultiModal versions of existing models
  - Assumption is normal, depths, and masking will improve classification
- DepthAnything also has some usage of the iphone TrueDepth cameras to “correct” the images produced
- Try again with the larger model when released:
  - Depth-Anything-V2-Giant 1.3B Coming soon
- Use PCA to extract orientation of detected object point clouds (eigenvector / eigenvalues)
- Use detected classes to fit cylinders to cups, or other primitives based on object classes to substitute for simulation.
  - ie: use normals to determine cup radius and fit cylinder on longest axis
  - Could use a CNN to determine if a cup is right side up or upside down
  - Or determine it from point clouds as principal axis can be backwards
  - RANSAC algorithms may help

RELATED WORK

- Gaussian Splatting -- https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/
  - Takes the point cloud and makes each point a 3d gaussian (ellipsoid) with alpha value
  - Use gradient descent to reduce error between photos and rendered viewpoints by directly modifying gaussians
  - Uses COLMAP to solve for camera position similar to photogrammetry
- NeRF -- https://github.com/bmild/nerf
  - Neural Radiance Fields also using COLMAP to solve for camera positions
  - Predates Gaussian Splatting, but inspired it
  - Uses a neural network to map (x,y,z, yaw_angle, pitch_angle) to a radiance field
  - Kind of blurry, requires running neural training at runtime to improve quality

GITHUB LINK

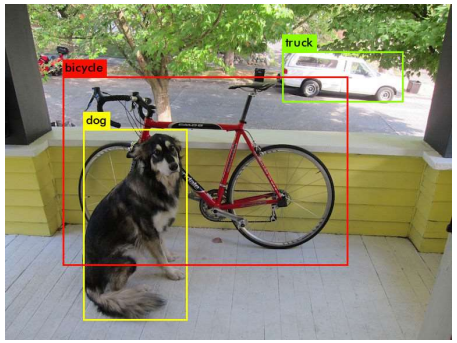
All code used and outputs on this repo:
http://github.com/awright2009/YOLO_misc/

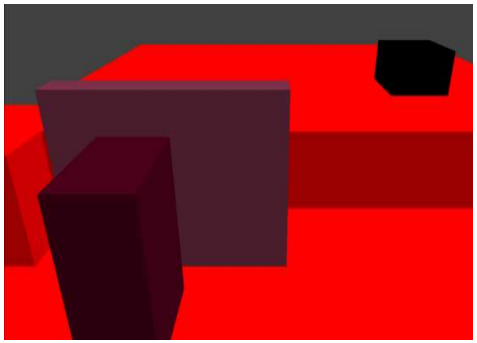
Videos will be concatenated and uploaded to youtube:
https://youtu.be/gnjmazjuQhY

MODELS USED

- YOLOv11 Ultralytics -- https://github.com/ultralytics/ultralytics
  - pip install ultralytics
  - Download weight and bias “checkpoints” separately
- SAM2 -- https://github.com/facebookresearch/sam2
  - pip install ultralytics
  - Download weight and bias “checkpoints” separately
- DepthAnythingV2 -- https://github.com/DepthAnything/Depth-Anything-V2
  - Download repo
  - pip install –e .
  - Download weight and bias “checkpoints” separately

ASPRIATION VS IMPLEMENTATION





ADDENDUM

- Differential Rendering / Inverse Rendering
  - https://jjbannister.github.io/tinydiffrast/
  - https://srush.github.io/DiffRast/

Takes traditional rasterization, replaces discontinuous pixels with continuous gaussians, and maintains autodiff (gradients) throughout the process, allows for back propagation.

Allows you to do gradient descent to correct the point cloud positions against reference images

ADDENDUM

Remember how we couldn't get the two rotated stereo depth images at 30° angles to correspond well with non-metric depth due to distortion?

Still a work in progress, but this should be the perfect job for gradient descent on a differentiable rasterizer

- PyTorch3D -- https://github.com/facebookresearch/pytorch3d
- NvDiffRast -- https://github.com/NVlabs/nvdiffrast
- TensorFlow3D -- https://research.google/blog/3d-scene-understanding-with-tensorflow-3d/

THANK YOU