Lecture 12:

Background: the light field and rendering basics

Visual Computing Systems
Stanford CS348K, Spring 2023
Many scene representations in graphics

Triangle-based 3D surface representations (mesh + surface materials)
(Rendering via ray-casting or 2D projection)


Depth-image based surface representations
(Novel view synthesis via depth-guided image warping, pixel re-projection, etc.)

3D Volumes

And many more... e.g., Implicit Surfaces

[Credit: Lee Griggs]
Novel view synthesis problem

Input photos (from a fixed set of views)

Novel views
(camera position different from those in input photos)
Fundamentals: the light field
Recall basic pinhole camera

Sensor plane: (X, Y)

Pixel P1

Pixel P2

Pinhole

Scene object 1

Scene object 2
What about taking the pictures from a new viewpoint?

Scene object 1

Scene object 2

Sensor plane: (X,Y)

Pixel P1

Pixel P2

Pinhole
Light-field parameterization

Light field as a 4D function (represents light in free space: no occlusion)

Efficient two-plane parameterization
Line described by connecting point on \((u,v)\) plane with point on \((s,t)\) plane
If one of the planes placed at infinity: point + direction representation

Levoy/Hanrahan refer to representation as a “light slab”: beam of light entering one quadrilateral and exiting another
Sampling the light field

Simplification: only showing lines in 2D
(full light field is 4D function)
Measuring the light field by taking many pictures

[U=1 --> S=1]  [U=0 --> S=0]
Stanford Camera Array

640 x 480 tightly synchronized, repositionable cameras

Custom processing board per camera

Tethered to PCs for additional processing/storage

[Wilburn et al. 2005]
Light field storage layouts

(a)

(b)

[Image credit: Levoy and Hanrahan 96]
Later light field cameras

Lytro Illum
Acquiring light field content for VR

Google’s Jump VR video:
Yi Halo Camera (17 cameras)

Facebook Manifold
(16 8K cameras)
Stereo, 360-degree viewing
Stereo, 360-degree viewing
Measuring light arriving at left eye

Left eye

\[ \sin \theta = \frac{r}{R} \]
Measuring light arriving at right eye

\[
\sin \theta = -\frac{r}{R}
\]
How to estimate rays at “missing” views?
Interpolation to novel views depends on scene depth
Interpolation to novel views depends on scene depth
Computing depth of scene point from two images

Binocular stereo 3D reconstruction of point $P$: depth from disparity

- **Focal length:** $f$
- **Baseline:** $b$
- **Disparity:** $d = x' - x$

$$z = \frac{bf}{d}$$

Simple reconstruction example: cameras aligned (coplanar sensors), separated by known distance, same focal length

“Disparity” is the distance between object’s projected position in the two images: $x - x'$
Microsoft XBox 360 Kinect

Illuminant
(Infrared Laser + diffuser)

RGB CMOS Sensor
640x480 (w/ Bayer mosaic)

Monochrome Infrared CMOS Sensor
(Aptina MT9M001)
1280x1024 **

** Kinect returns 640x480 disparity image, suspect sensor is configured for 2x2 pixel binning down to 640x512, then crop
Infrared image of Kinect illuminant output
Infrared image of Kinect illuminant output

Credit: www.futurepicture.org
Correspondence problem

How to determine which pairs of pixels in image 1 and image 2 correspond to the same scene point?
Correspondence problem = compute “flow” between adjacent cameras

- For each pixel in frame from camera $i$, find closest pixel in camera $i+1$
- Google’s Jump VR video pipeline uses a coarse-to-fine algorithm: align 32x32 blocks by searching over local window, then perform per-pixel alignment
  - Recall: H.264 motion estimation, HDR+ burst alignment (same correspondence challenge, but here we are aligning different perspectives at the same time to estimate unknown scene depth, not estimating motion of camera or scene over time)
  - Additional tricks to ensure temporal consistency of flow over time (see papers)
Left eye: with interpolated rays

[Credit: Camera icon by Venkatesh Aiyulu from The Noun Project]
“Casual 3D photography”

- Acquisition: wave a smartphone camera around to acquire images of scene from multiple viewpoints
- Processing: construct 3D representation of scene from photos
  - Render a textured triangle mesh

Dual-camera Smartphone
Burst of photos + depth maps
Stitch photos into depth panorama, create 3D mesh + textures, render during VR viewing
But it’s hard to estimate depth or geometry

View out my window in Gates
Volumetric representations

\[ \sigma(p) \]

\[ c(p, \omega) = c(x, y, z, \phi, \theta) \]

Volume density and color at all points in space.
Representing rays

$r(t) = o + td$

- origin
- unit direction
- point along ray
- "Distance" or "time"
Absorption in a volume

\[ L(p, \omega) \xrightarrow{\sigma_a(p)} L + dL \]

\[ dL(p, \omega) = -\sigma_a(p) L(p, \omega) \, ds \]

- \( L(p, \omega) \): light energy (radiance) along a ray from \( p \) in direction \( \omega \)
- Absorption cross section at point in space: \( \sigma_a(p) \)
  - Probability of being absorbed per unit length
  - Units: 1/distance

\[ p = (x, y, z) \]
\[ \omega = (\phi, \theta) \]
Rendering volumes

\[ \sigma(p) \]
\[ c(p, \omega) \]

Volume density and color at all points in space.
e.g., Values stored in a 3D grid

\[ C(r) = \int_{t_n}^{t_f} T(t) \sigma(r(t)) c(r(t), d) dt, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s)) ds\right) \]
Regular 3D grid representation?

Consider storage requirements:
$1024^3$ cells

Ignore directional dependency: $\text{rgb} \sigma$ 4 bytes/cell
(\sim 4 \text{ GB})

Now consider directional dependency on $(\phi, \theta)$
... much worse

Typical challenge: limited resolution
Learning (compressed) representations

Why not just learn an approximation to the continuous function that matches observations from different viewpoints?

\[(p, \omega) \rightarrow F_{\theta}(p, \omega) \rightarrow \sigma(p) c(p, \omega)\]
Learning better (more compressed) representations

- Why not just learn an approximation to the continuous function:

\[(p, \omega) \rightarrow F_\theta(p, \omega) \rightarrow \sigma(p) \rightarrow c(p, \omega)\]

- For all photos of the scene that we have, use \(F_\theta(p, \omega)\) to volume render the scene from the known viewpoint.

- Loss is difference between rendered view and actual photo.

- Update \(\theta\) using standard optimization techniques (SGD)
Learning neural radiance fields (NeRF)
What just happened?

- Continuous coordinate-based representation vs regular grid: MLP “learns” how to use its weights to produce high-resolution output where needed... given input data

- Compact representation: trades-off space for expensive rendering
  - Good: a few MBs = effectively very high resolution dense grid
  - Bad: must evaluate MLP every step
    - And it’s a “big” MLP (8-layer x 256)
  - Bad: must step densely (because we don’t know where the surface is)

- Compact representation: optimization can learns to interpolate views despite complexity of volume density and radiance function
  - Only structural bias is the separation into positional \( \sigma \) and directional rgb
  - Training time: hours to a day to learn a good NeRF
Demos
Key ideas of volumetric representations in this context

- Do not need to reconstruct/estimate triangle mesh surface geometry
- Volume rendering is easily differentiable, so easy to update $F_\theta(p, \omega)$
- The DNN used to represent $F_\theta(p, \omega)$ is a compact representation of this high-dimensional function.
  - Better representation than a dense voxel grid.