Lecture 17:

Basic video conf + Differentiable rendering to reconstruct 3D scenes

Visual Computing Systems
Stanford CS348K, Spring 2025

Anticipated project "results" thinking activity

- Step 1: In 2-3 sentences… what is your project's goal, and what is your definition of success? I want you to be as specific as possible.
  - Key phrases for the speaker: "I hope to show that…", "I hope to make…"
  - A key phrase for the listener: "What do you mean by _____? Did you mean ____ or ____?

Anticipated project "results" thinking activity

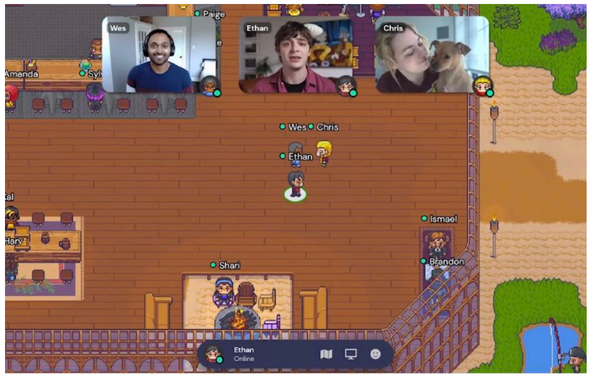
- Step 2:
  - What is the primary technical challenge / design challenge / hurdle that you have to overcome or learn? OR… What is the primary technical question you are trying to answer?
    - Phrases for the speaker: The primary challenge of the whole project is _____… The reason why it's hard is _____.. The think my partners and I figuring out is _____…
    - Phrases for the listener: "What do you mean by _____? Did you mean ____ or ____?

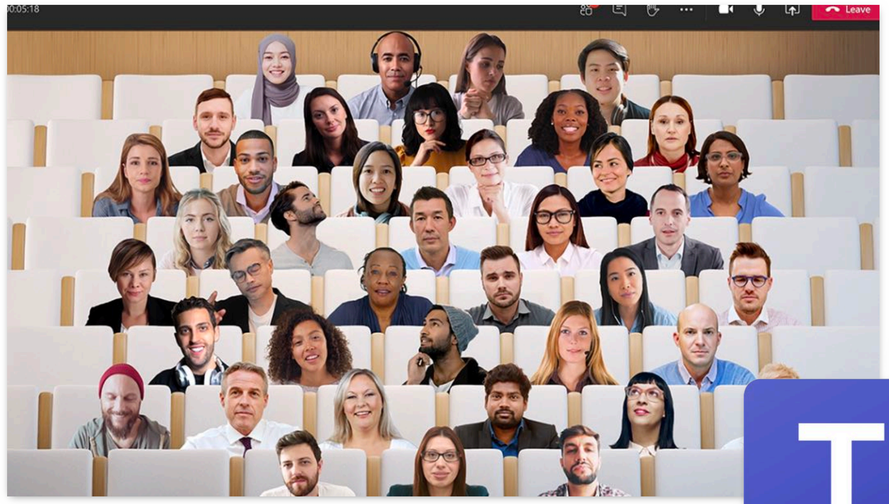
Project “results” thinking activity

- Step 3: To show I was successful I am going to show this result/demo/graph. My prediction is that I will see this _____. But if I see _____ that’s a sign I was not successful.
  - Listener: do you agree that it would convince you success would be achieved? If not, what would you prefer to see instead?

Part I:
Videoconferencing systems
(Very quickly)

As you can imagine, a lot of interest in video conferencing

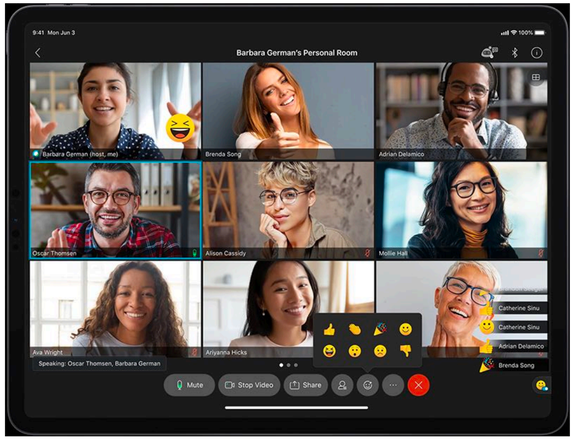




zoom

Gather

BlueJeans



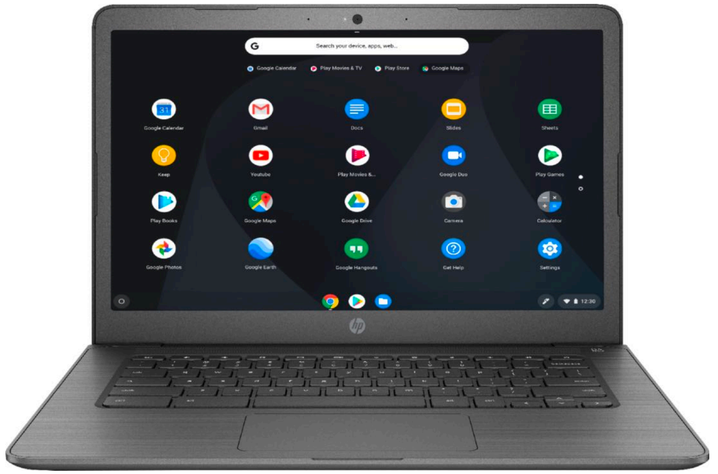
nooks

Webex Meetings

Let’s design a video conferencing system

We want to deliver a visually rich experience similar to features of modern platforms

Deliver to wide range of clients and network settings



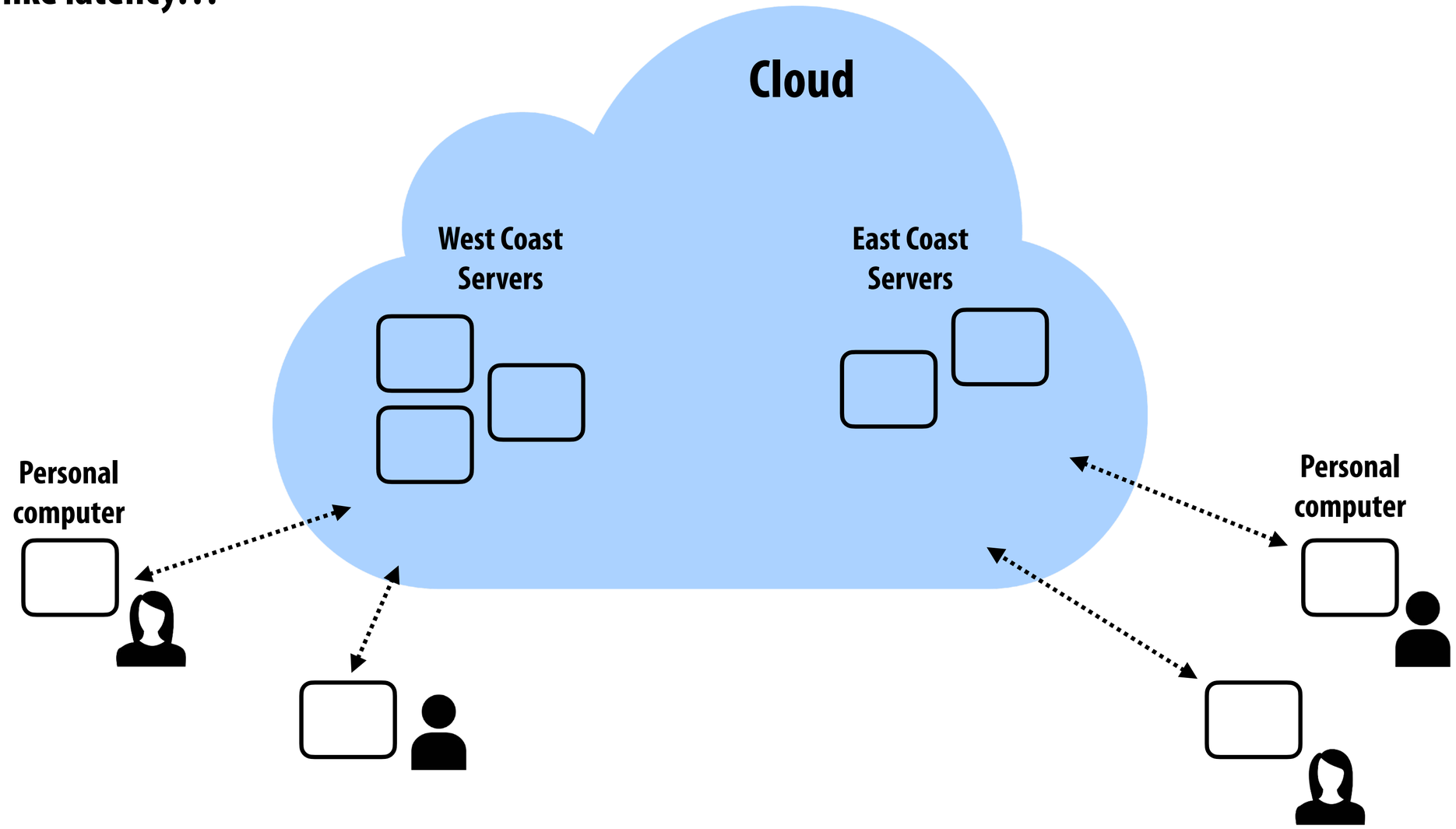
Let's design a video conferencing system

Large gallery views: companies raced to provide 7x7 gallery in 2020 *

*** it was quickly determined this was not particularly great feature**

Setup…

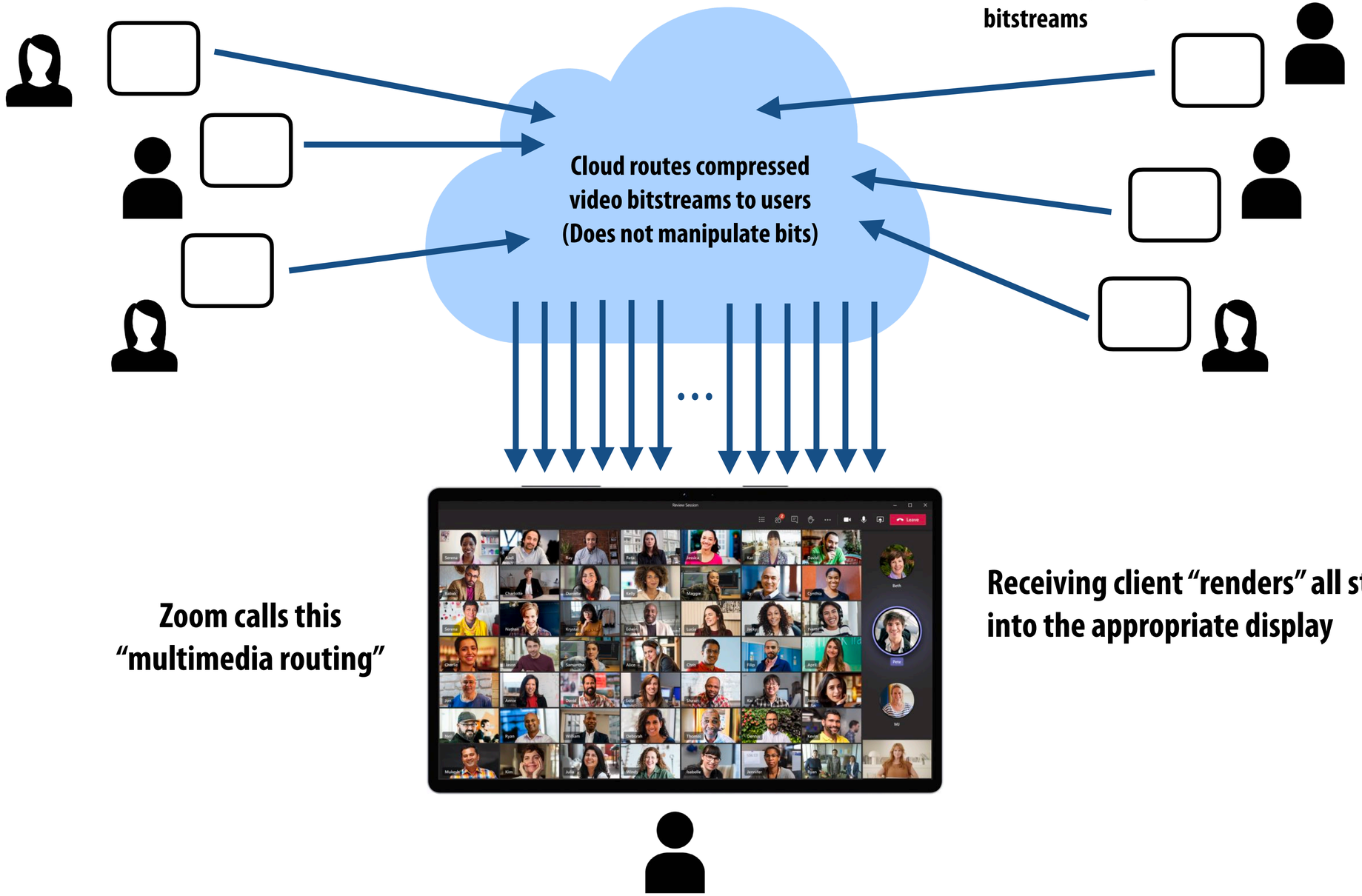
Consider issues like latency…



Q. Should we transcode/process video on our cloud servers?

- What are advantages (to users? To the service provider)?
- What are disadvantages?

Implementing gallery view



One drawback of this design

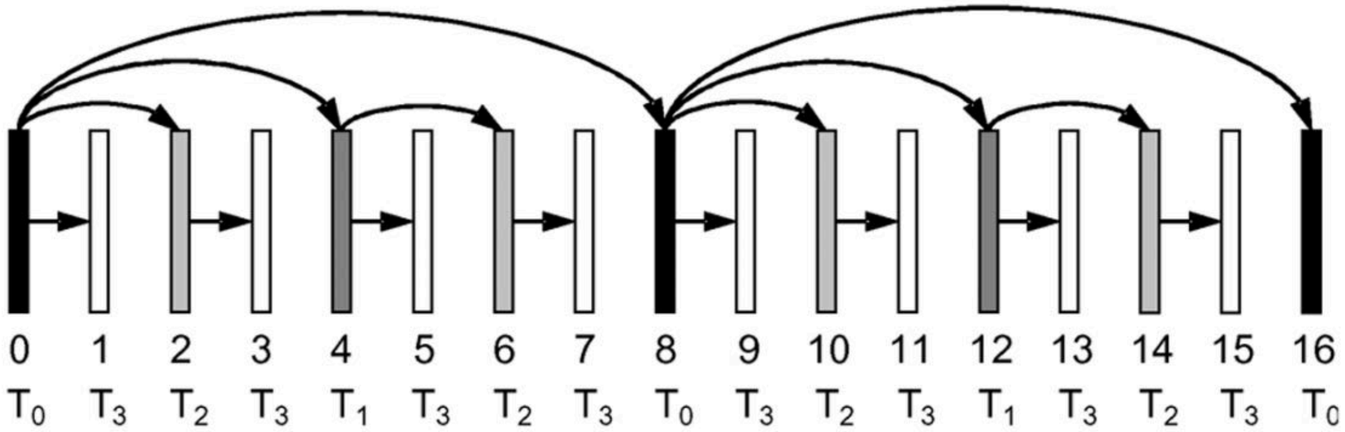
- If each client is providing a single compressed video stream, that means each person on the video call must receive the same bits right? (What if they are on different network connections?)

Scalable video codec (SVC)

SVC is an extension of H.264 standard

- "Scalable" compressed video bitstream: subsets of the bitstream encode valid video streams for a decoder
  - Implication: if packets get lost, the remaining packets form a valid H.264 bitstream, albeit at lower resolution or quality

Example: temporal scalability



Layer 0: (T_0) defines valid video at frame rate R
Layer 1 (T_1) defines bumps frame rate to 2R
…

Note that layer 0 information is used to predict higher layer information (see arrows)

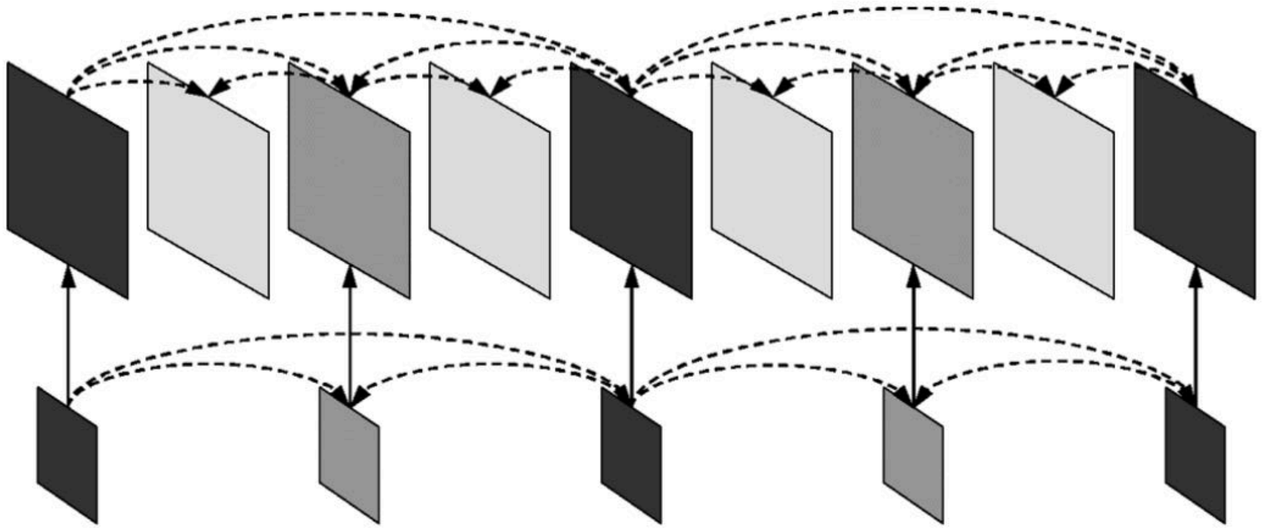
Scalable video codec (SVC)

SVC is an extension of H.264 standard

Example: spatial scalability

Layer 1:
(Higher res)

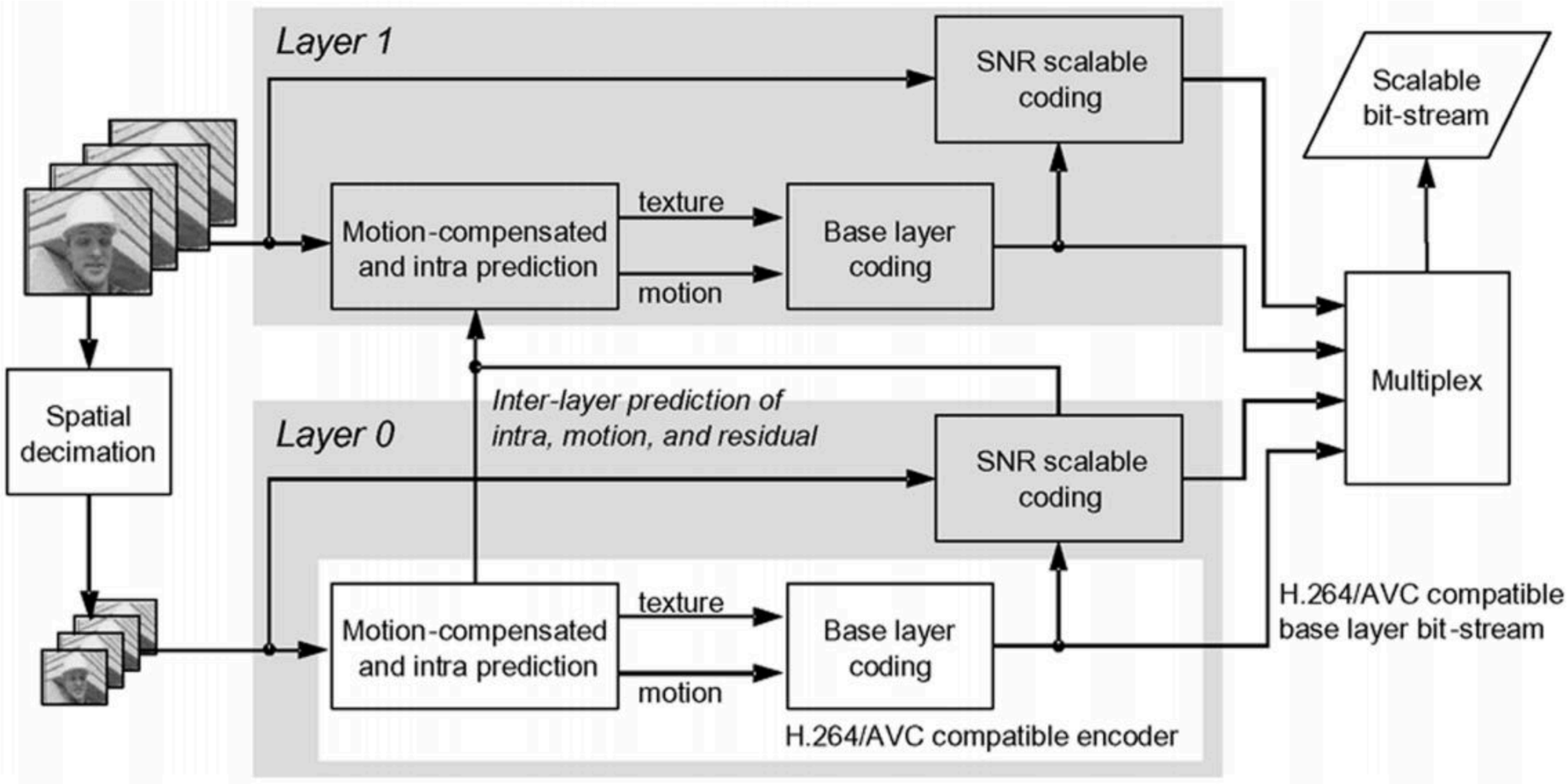
Layer 0:
(Low res)



Again, note how layer 0 information is used to predict higher layer information
(Higher efficiency than independently encoding two video streams)

Layer 0: defines valid video at low resolution (and low frame rate)
Layer 1: provides additional information for higher resolution (and higher frame rate) video

Scalable video codec (SVC) encoder



Costs: higher encoding/decoding costs

(But possible on modern clients as SVC is supported in hardware)

Part II:
The value of differentiable rendering

(Applications of reconstructing 3D scene representations from images)

Last time: rendering volumes

Given "camera ray" from point o in direction w….

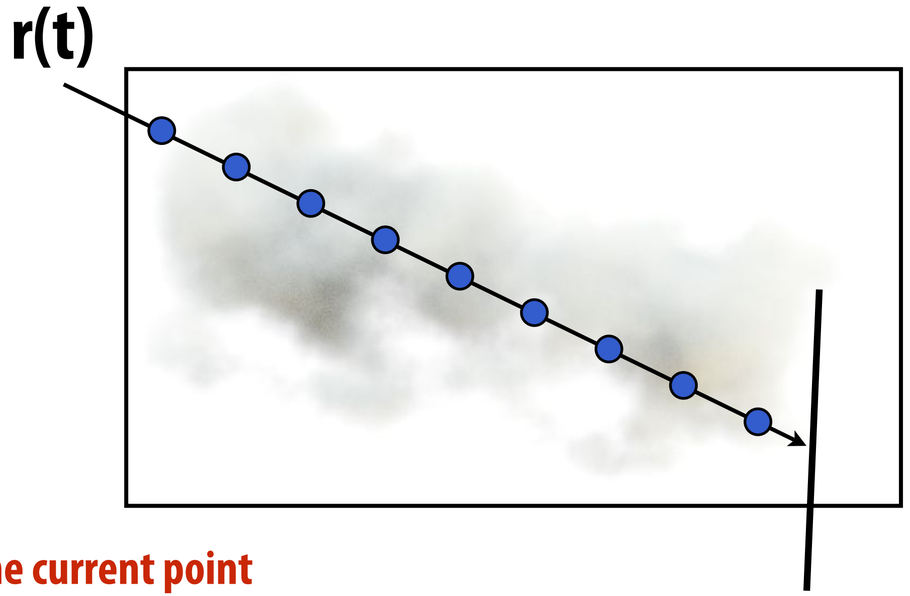
$$\mathbf{r}(t) = \mathbf{o} + t\omega$$

And continuous volume with density and directional radiance.

$\sigma(\mathrm{p})$

$c(\mathrm{p}, \omega)$

← Volume density and color at all points in space.



Step through the volume to compute radiance along the ray.

Attenuation of radiance along r between r(s) and "camera" due to out scattering or absorbion

Color, opacity of the volume at the current point
(More precisely: radiance along r at point r(s) due to in-scattering or emission)

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

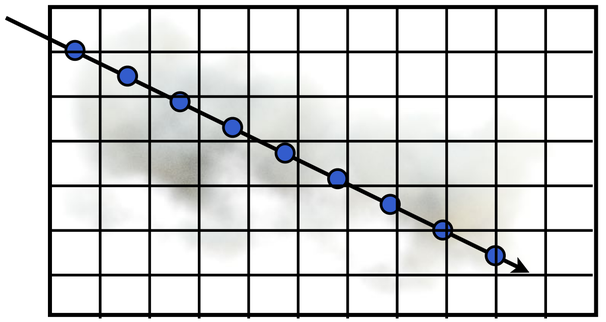
Recovering a volume that yields acquired images

Given a set of images of a subject with known camera positions…

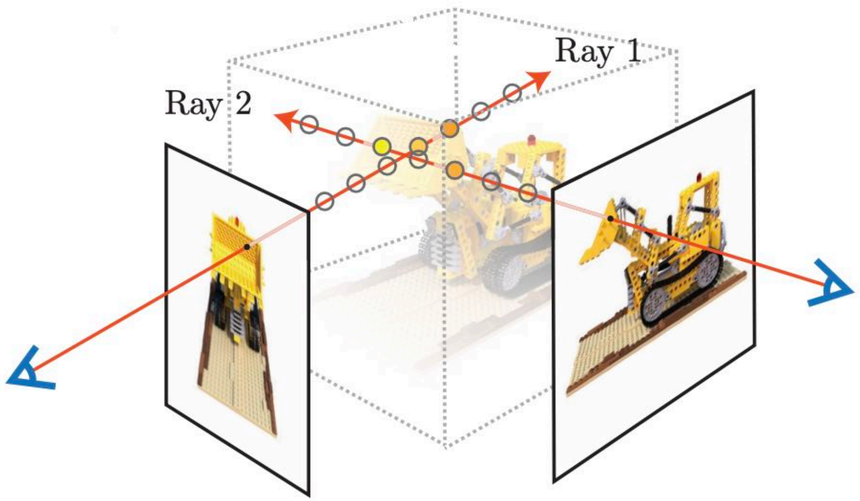
$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Idea: find volume parameters (opacity and color at each (i,j,k)
To make C(r) match the corresponding pixel in the photos.

r(s)

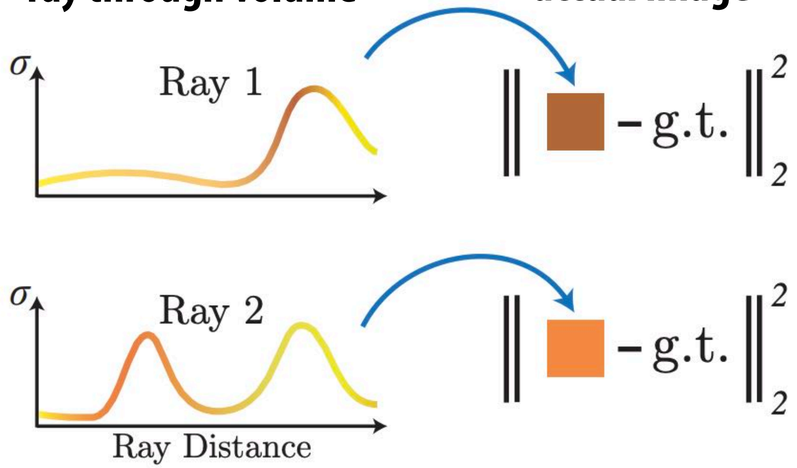


For many rays…. trace through volume… see if the result matches the photo… use error to update volume's opacity/color values

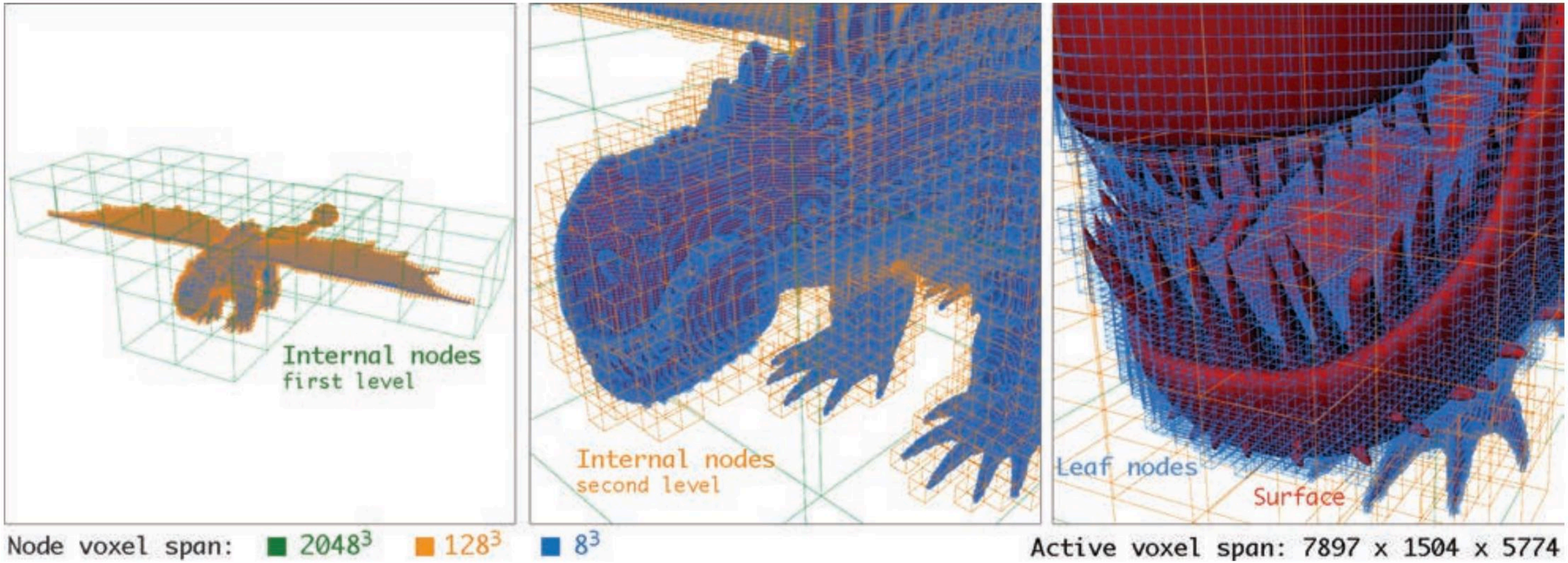


Compute radiance along ray through volume

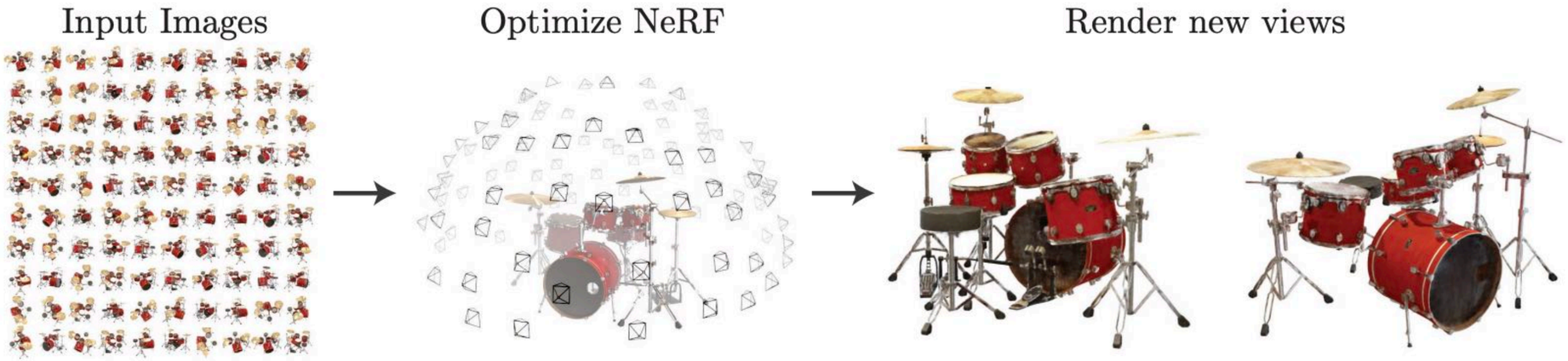
Compare to actual image

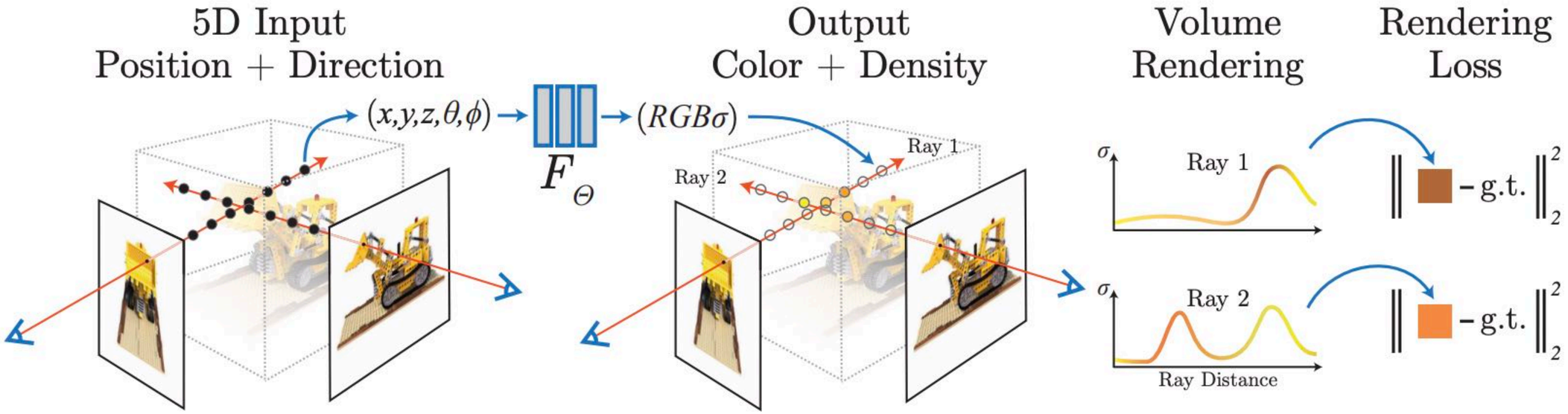


Sparse voxel node representation



Learning neural radiance fields (NeRF)





Key idea: differentiable volume renderer to compute dC/d(color)d(opacity)

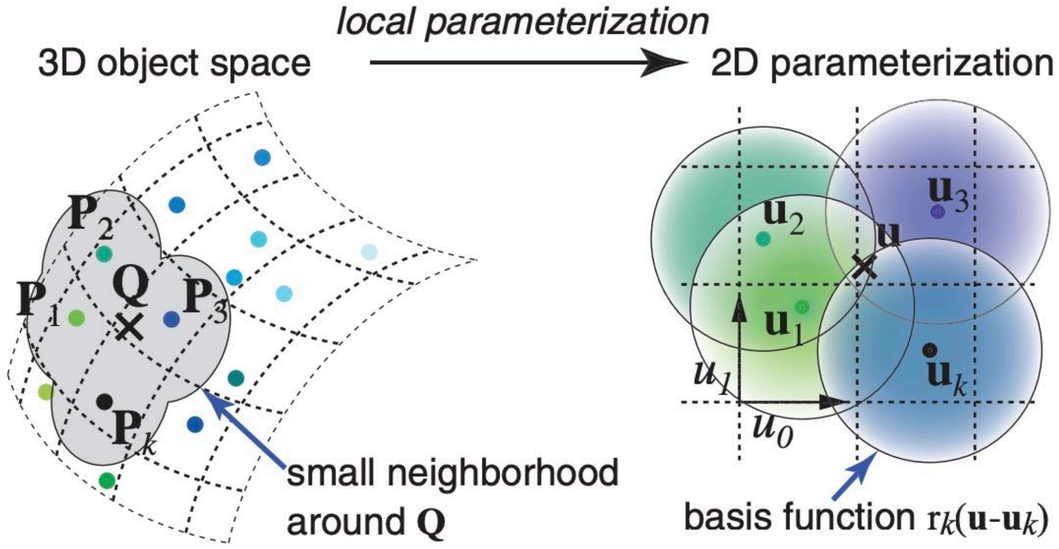
Great visual results!

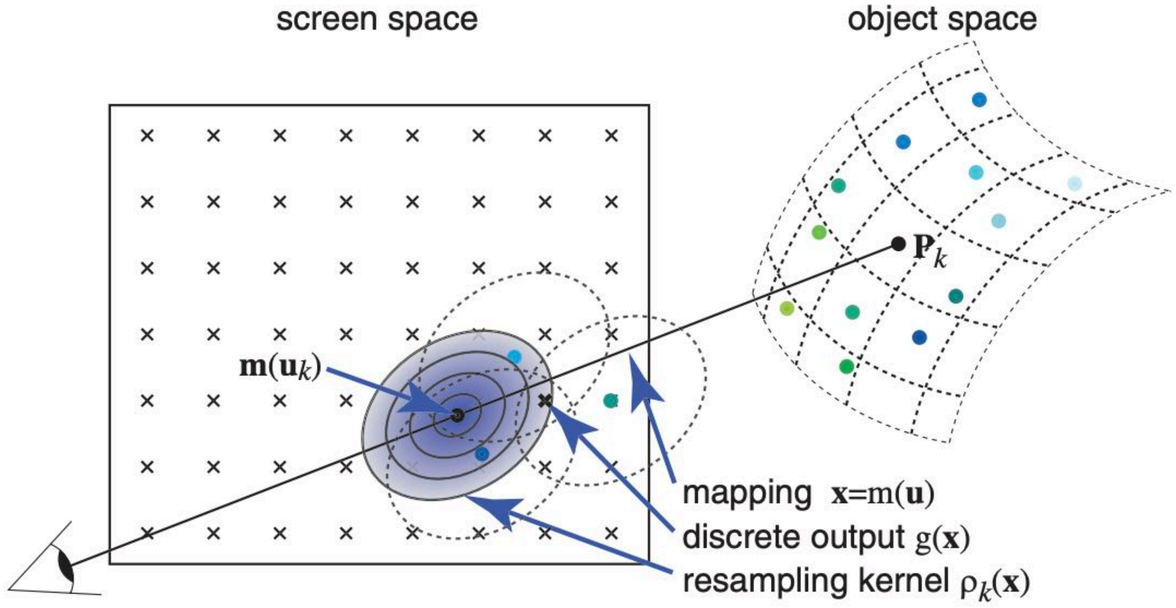
Credit: Mildenhall 2023

Rendering point clouds

Anti-aliasing point clouds

- Treat surface as a collection of "3D Gaussian blobs" (convolve points with Gaussian filter)





- 3D Gaussians turn into oriented 2D gaussians when projected onto the 2D screen

- Can render the blobs by rasterizing them back to front (this requires alpha compositing)

[Zwicker 2001]

Visualization of 3D Gaussians

Rendered Result

Visualization of 3D Gaussians

[Credit: Kerbl 2023]

“Gaussing splatting”

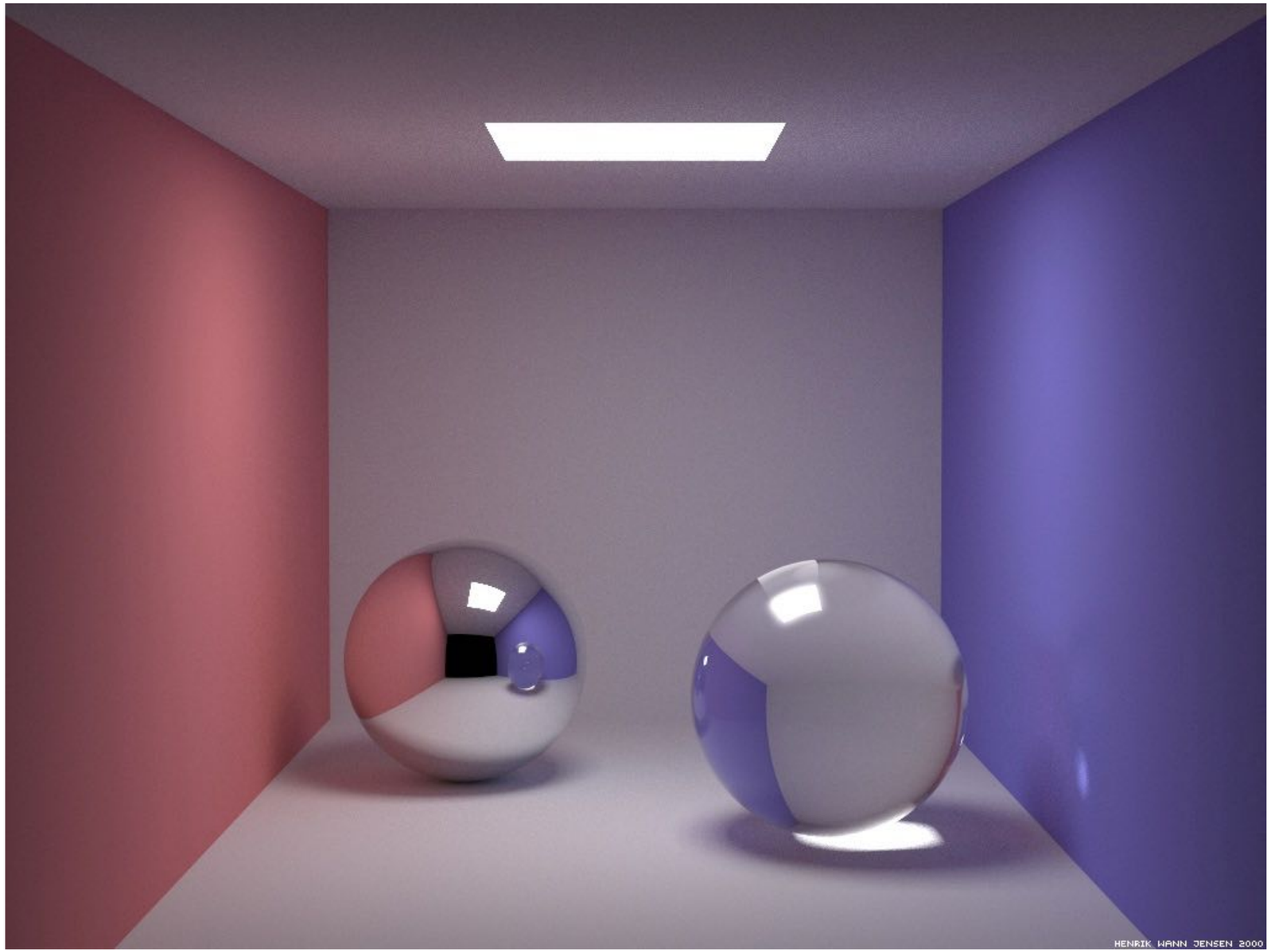
[Credit: Kerbl 2023]

What about more advanced representations?

- But in graphics we have much richer models of the visual world than just density at a point, or a set of colored gaussians.

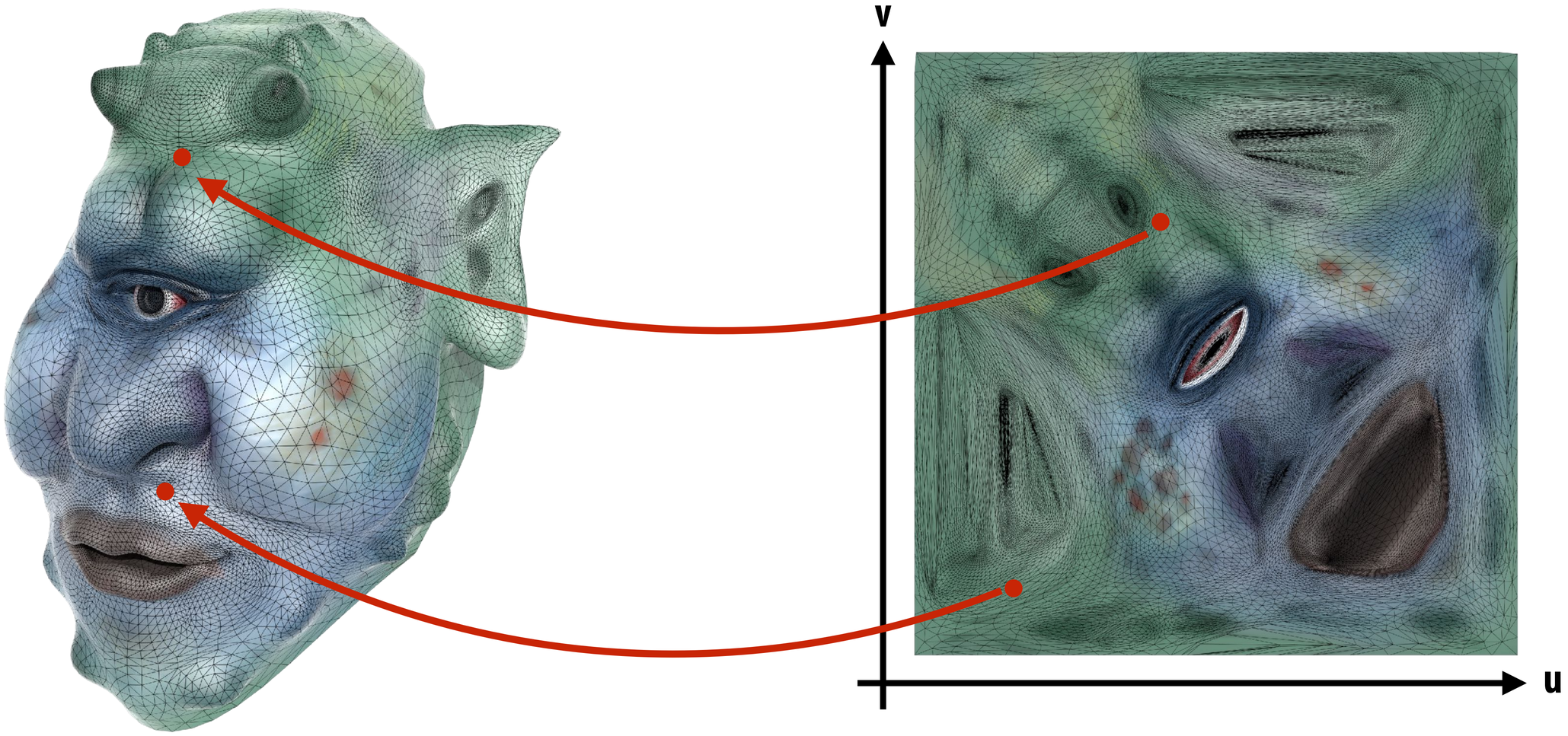
- For example, surfaces with complex material properties.

- Parameters describing surfaces (triangles), lines, curves:



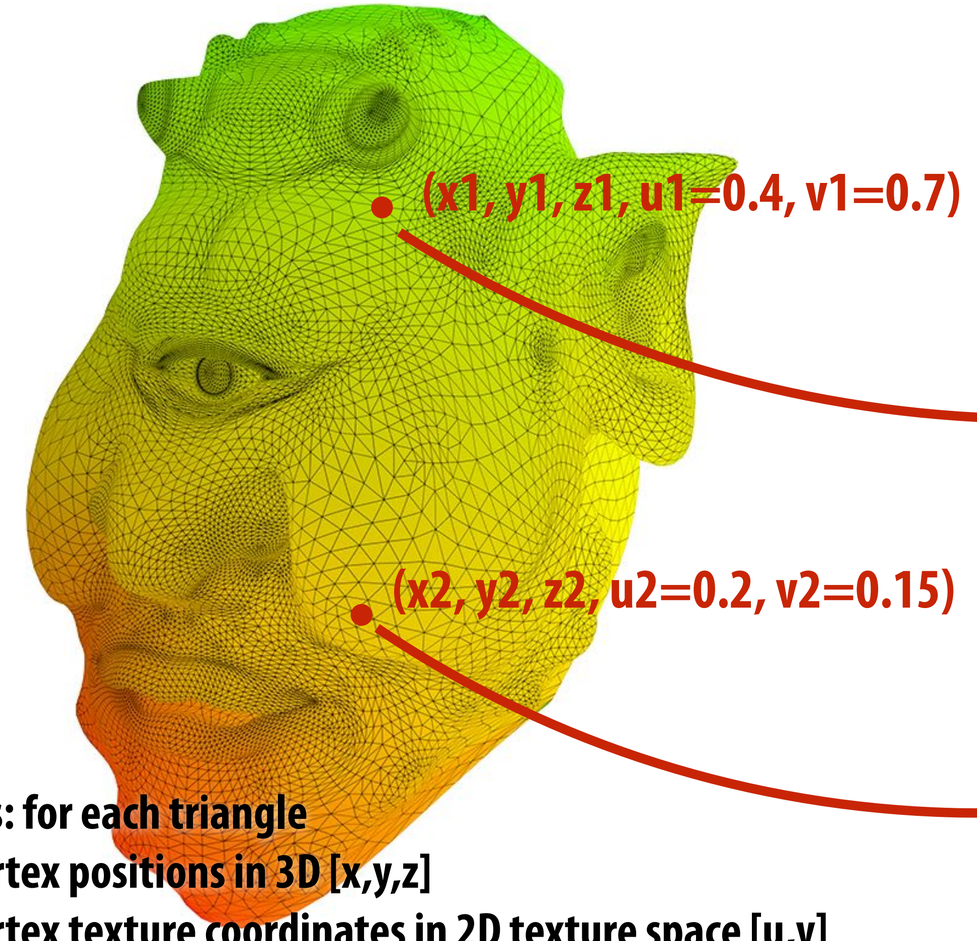
Example: texture as an efficient way to represent detail

Sample texture map at specified location in *texture coordinate space* to determine the surface's color at the corresponding point on surface.

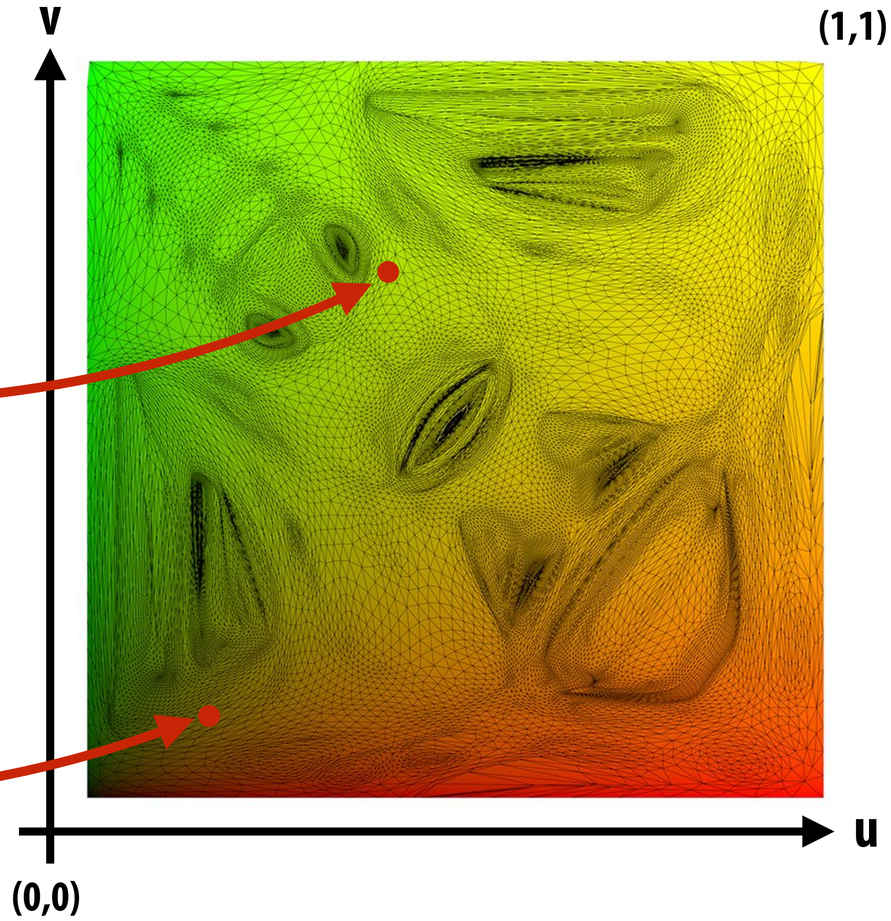


Every point on surface has a corresponding point in texture space

Visualization of texture coordinate value on mesh
(texture coordinate = color)



Visualization of location of triangle vertices
in texture space



Mesh inputs: for each triangle

- Per-vertex positions in 3D [x,y,z]
- Per-vertex texture coordinates in 2D texture space [u,v]

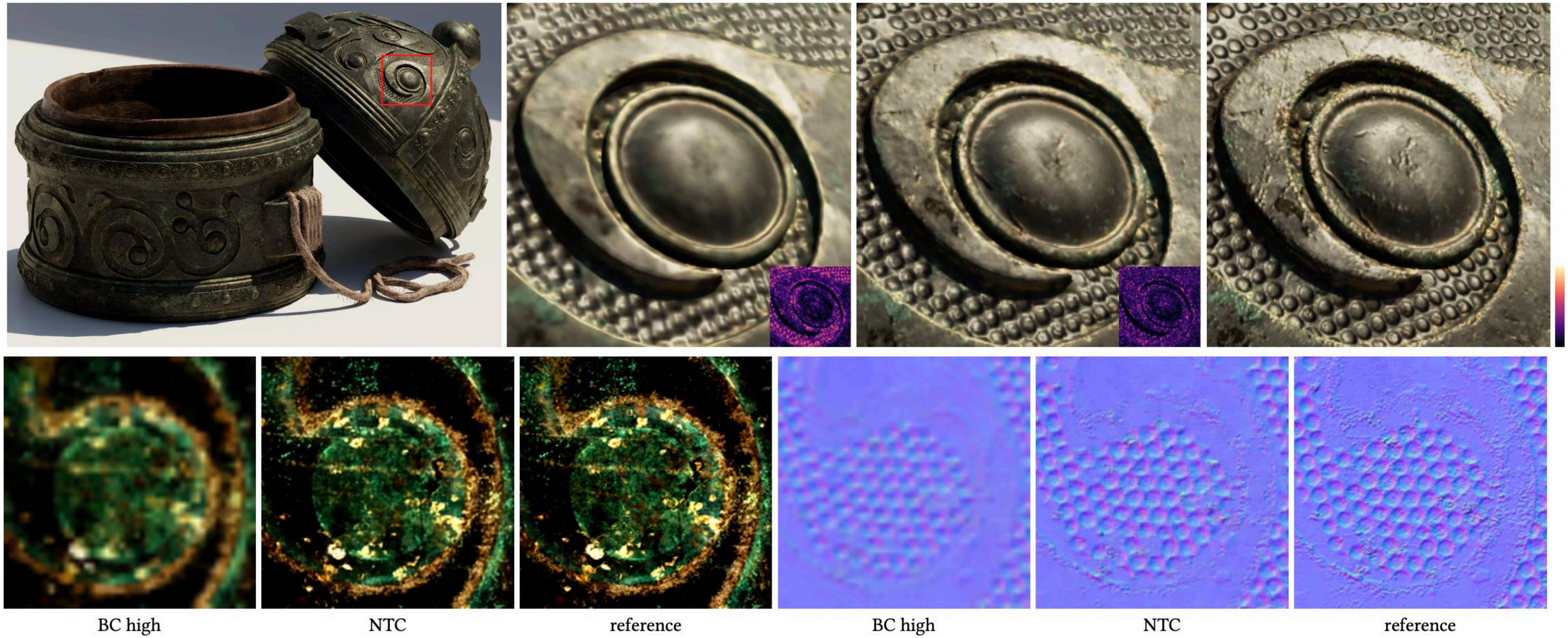
A full texture sampling operation to compute surface color

1. Compute u and v from screen sample x,y (via evaluation of attribute equations)
2. Compute du/dx, du/dy, dv/dx, dv/dy differentials from screen-adjacent samples.
3. Compute mip map level *d*
4. Convert normalized [0,1] texture coordinate (u,v) to texture coordinates U,V in [W,H]
5. Compute required texels in window of filter
6. Load required texels from memory (need eight texels for trilinear)
7. Perform tri-linear interpolation according to (U, V, d)

Takeaway: a texture sampling operation is not just an image pixel lookup! It involves a significant amount of math. Of which a gradient needs to be computed.

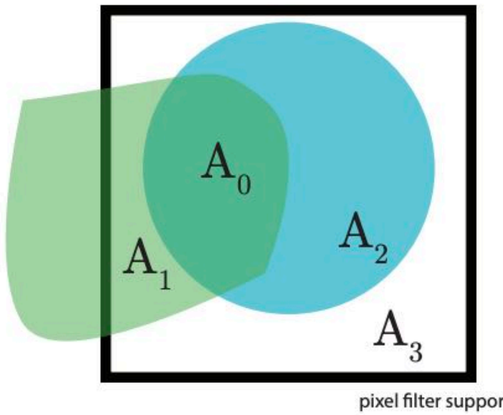
Neural texture compression

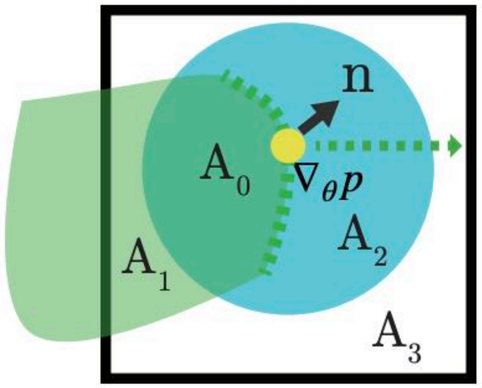
- Recall last week: learned compression schemes
- Learning parameters of the texture map… but encode as a DNN



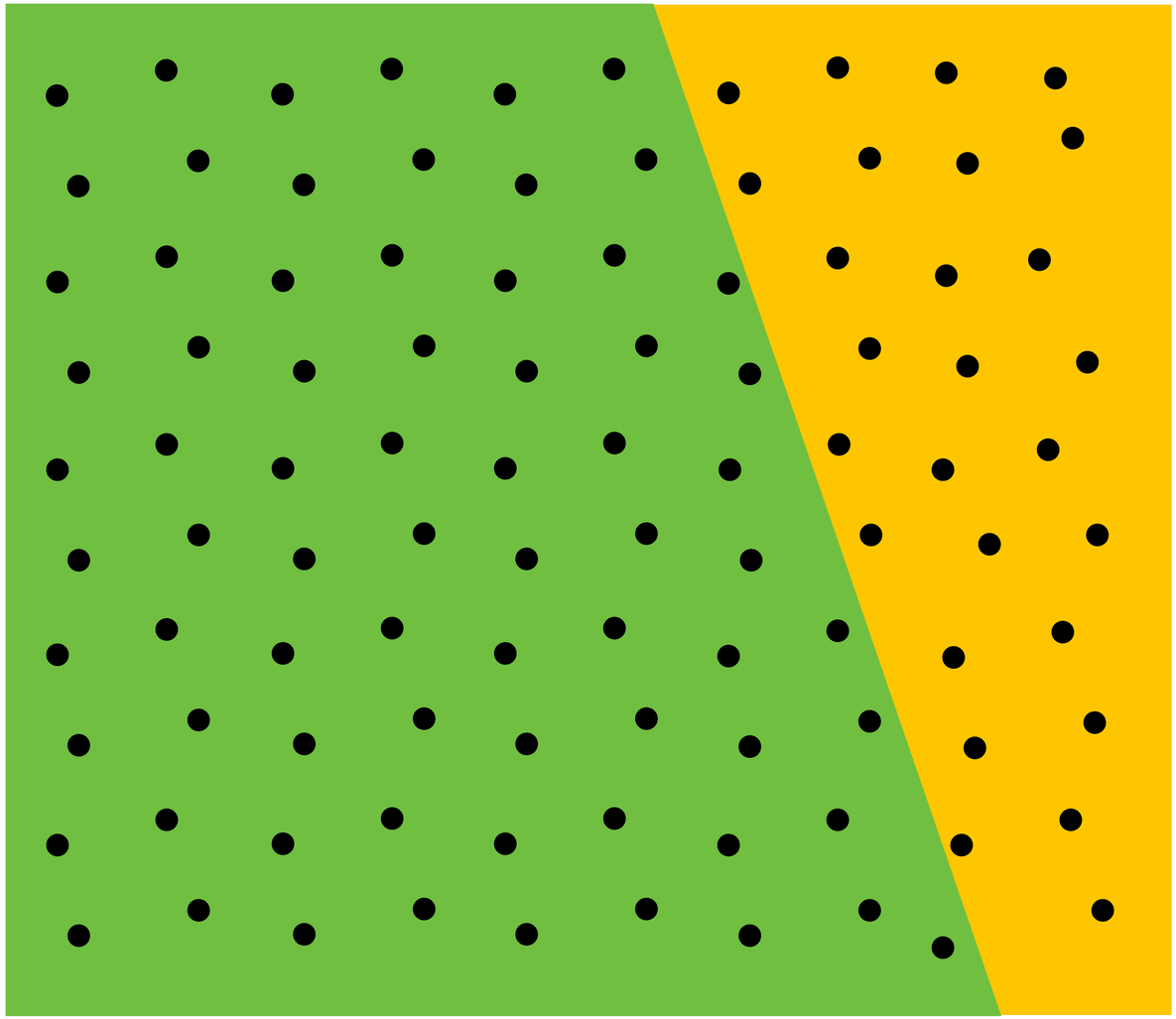
Gradients for textured triangle meshes?

- What are the parameters of a mesh? (Vertex positions, number of vertices, connectivity, etc.)
- Computing the gradient of a rendering subject to these parameters is challenging.
  - Consider simple case of fixed vertex count and fixed topology: the change in rendering output at a single sample point is discontinuous at object silhouettes as a function of vertex position changes (might see object A, then see object B if object A moves!)
  - But integral of radiance (aka color) over a pixel (post resolve output) is not discontinuous… (fraction of pixel covered)



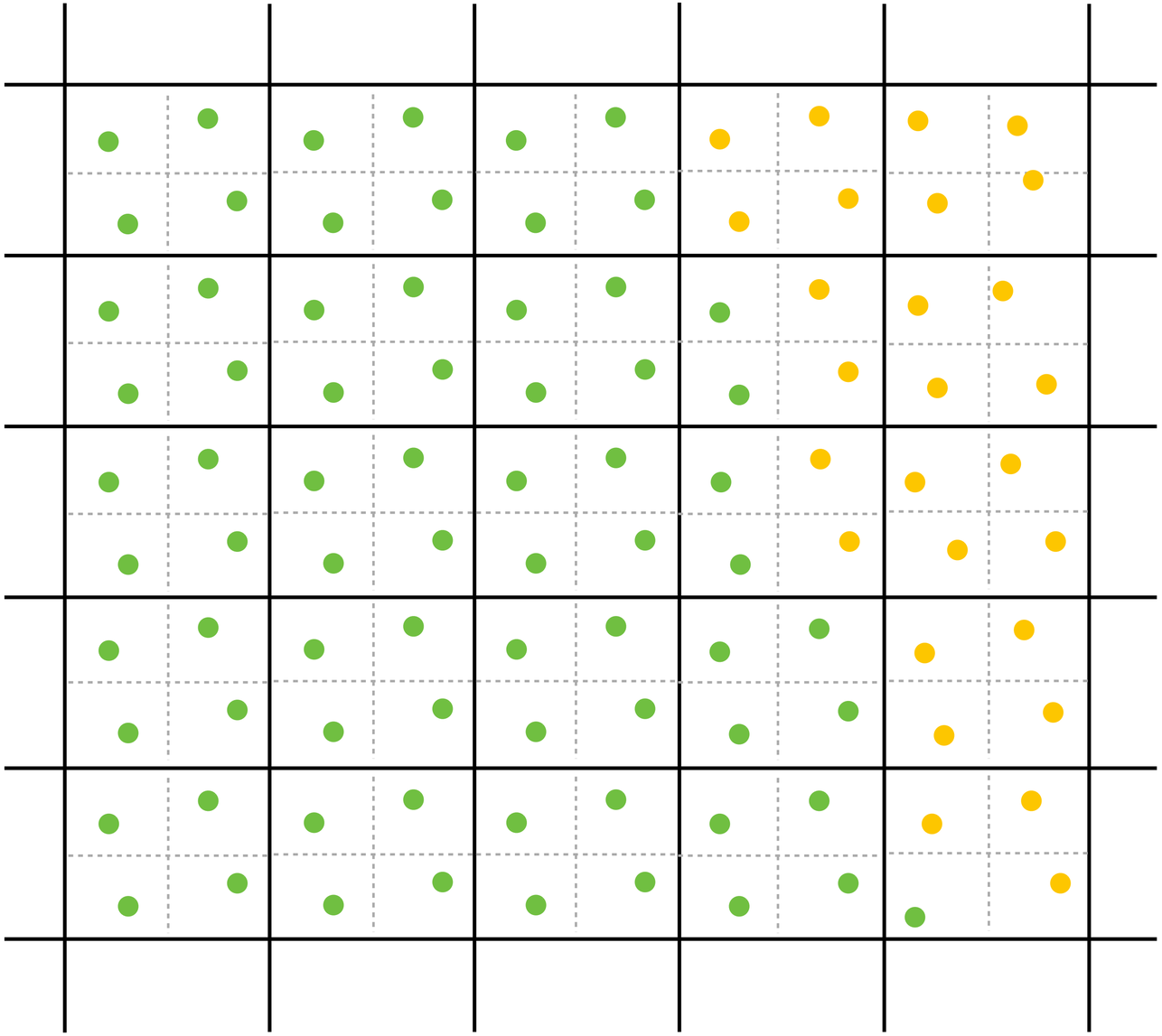


Consider rendering two surfaces

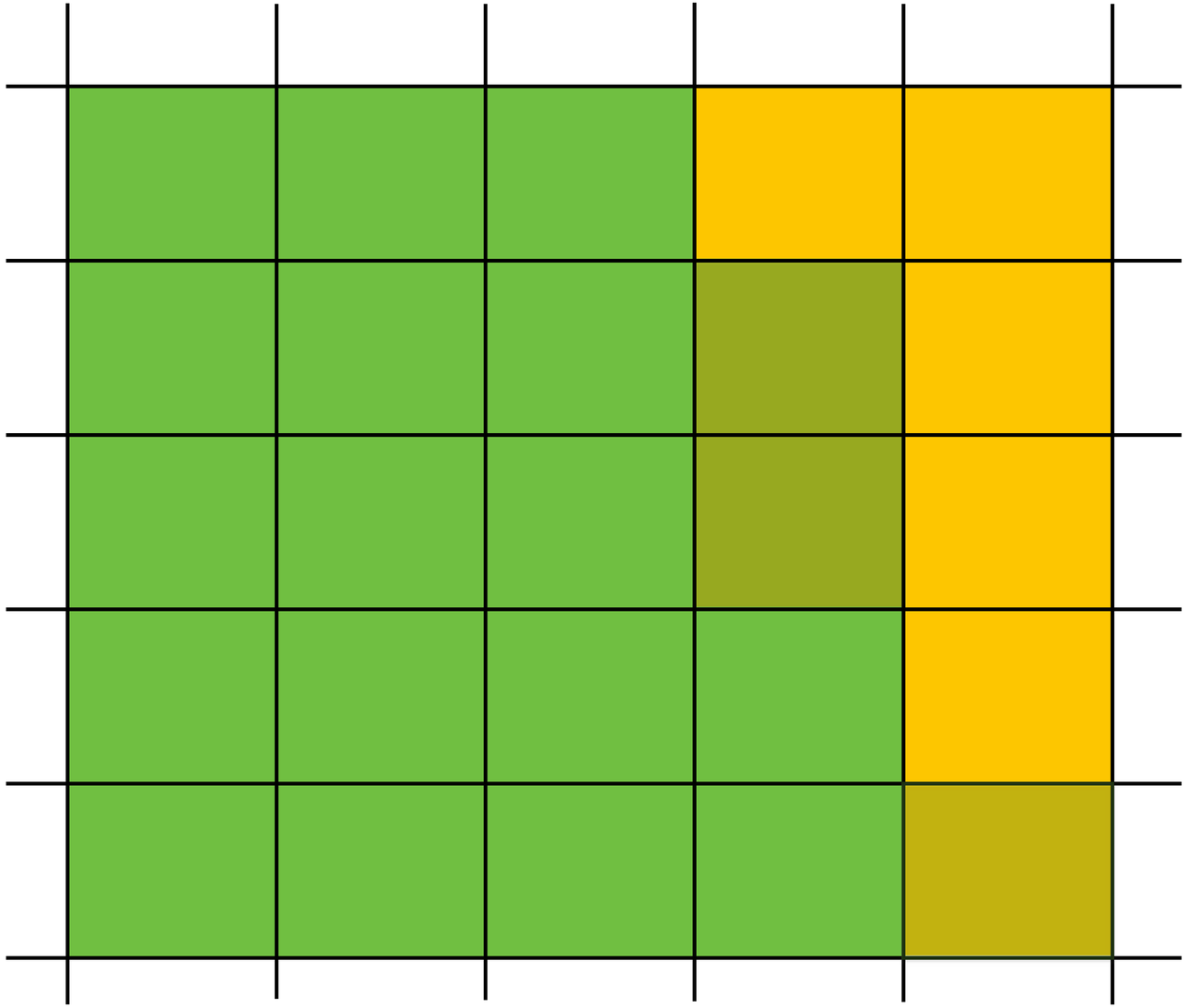


This example: green triangle occludes yellow triangle

Color buffer contents (4 samples per pixel)



Final resampled result



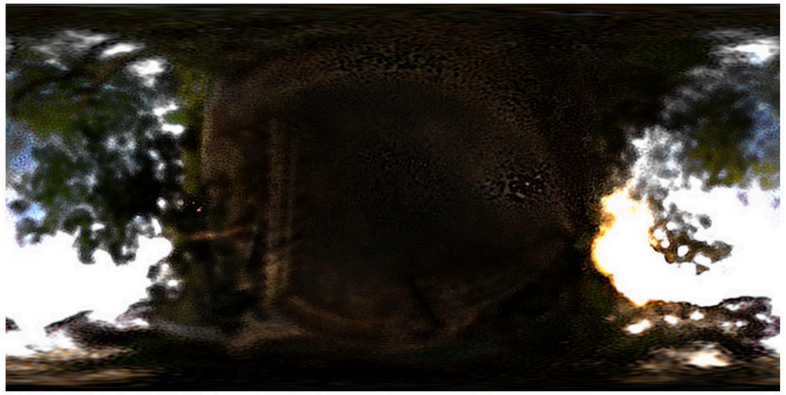
Note anti-aliasing of edge due to filtering of green and yellow samples.

Extracting meshes / materials / texture / lighting

Our shaded model

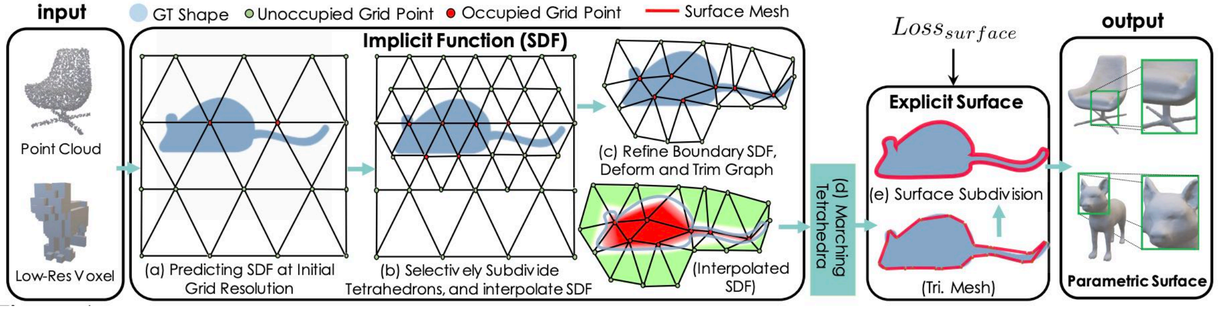
Reference image

Mesh /$\boldsymbol{k}_d$/$\boldsymbol{k}_{\text{orm}}$/$\mathbf{n}$



Extracted probe

Marching Tetrahedra algorithm to extract mesh

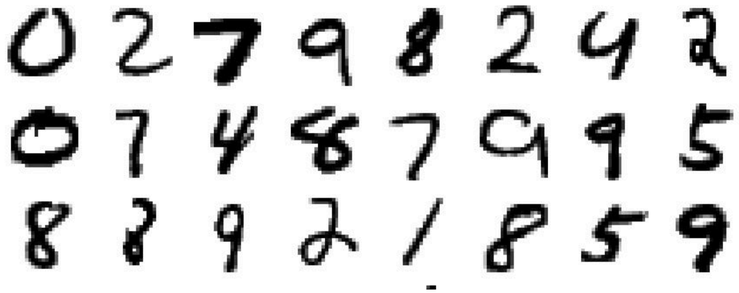


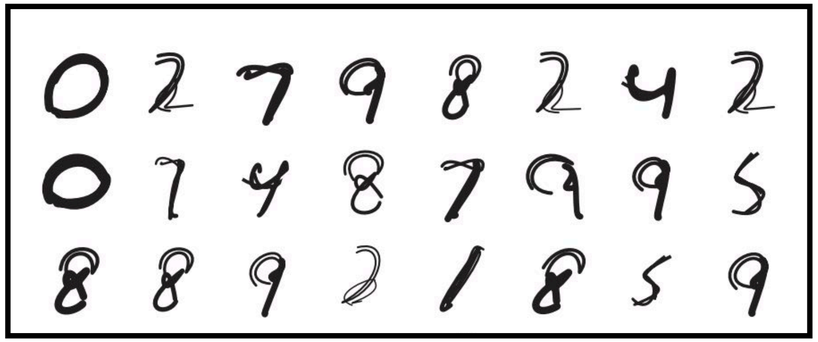
Example uses of differential rasterizers/ray tracers

- **Optimize parameters of SVG file to get a certain look**

Optimize "bold" parameter of SVG text to match image to right…

Optimize curve control points to match images of numbers.

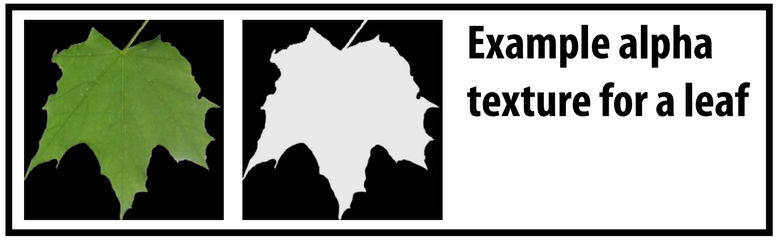




[Li et al. 2020 Differentiable Vector Graphics Rasterization for Editing and Learning]

Example uses of differential rasterizers/ray tracers

- Optimize vertex positions (at fixed vertex count) and also texture map pixels (alpha matte) to make the best low-poly representation of a mesh (when compared to renderings of a reference high poly mesh)



Initial guess (6.5k tris)

Optimized parameters

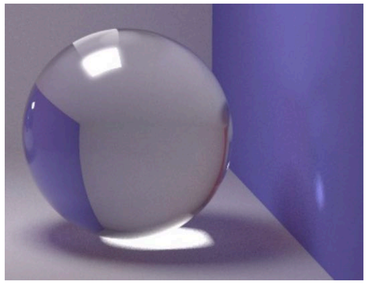
Our (6.5k tris)

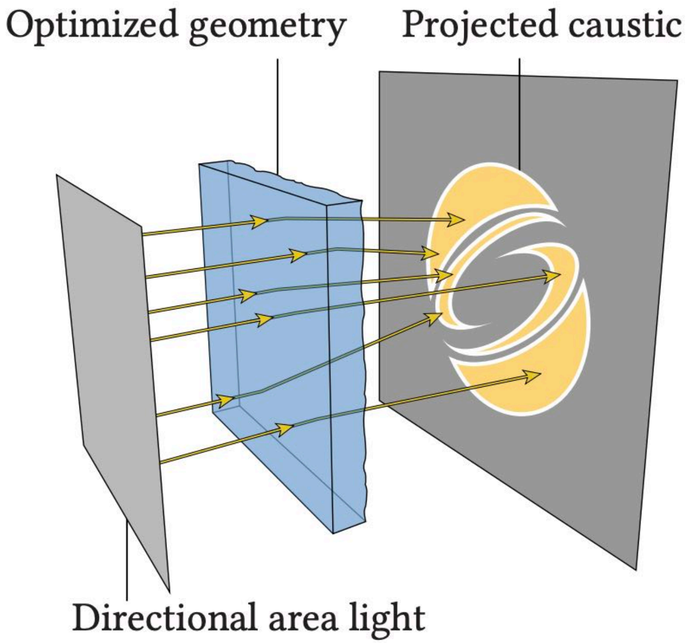
Reference (1.7M tris)

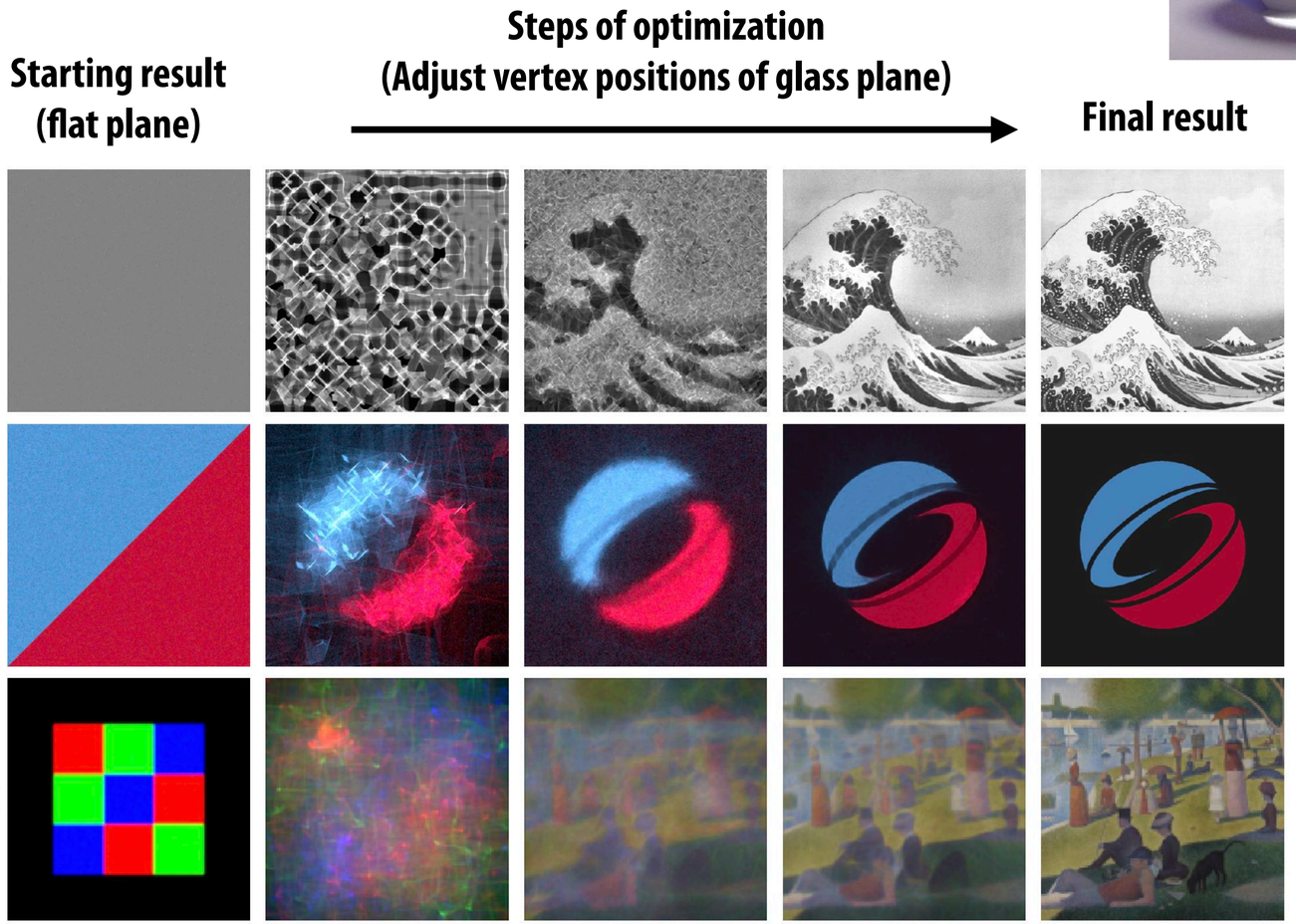
[Hasselgren et al. 2021 Appearance Driven automatic 3D Model Simplification]

Example uses of differential rasterizers/ray tracers

- "caustics" occur when refraction causes light to focus
  - Use differential renderer to optimize vertex positions so surface refracts light to make given image on a receiving plane.







[Nimier-David et al. 2019 - Mitsuba 2: A Retargetable Forward and Inverse Renderer]

Takeaways

- Diverse set of methods, data structures and functions used to represent scenes
  - Significant interest in estimating the parameters of these representations

- But now computing gradients is more than just differentiating a fixed-set of PyTorch primitives
  - Differentiating through a user's custom shader functions in a renderer
  - Differentiating through rasterization/ray tracing of complex scenes (such as triangle meshes)

- Need a general purpose language for writing GPU renderer code, and leveraging the compiler to generate gradients…