V: Polygonal Meshes & Rendering

3D CV

Kirill Struminsky

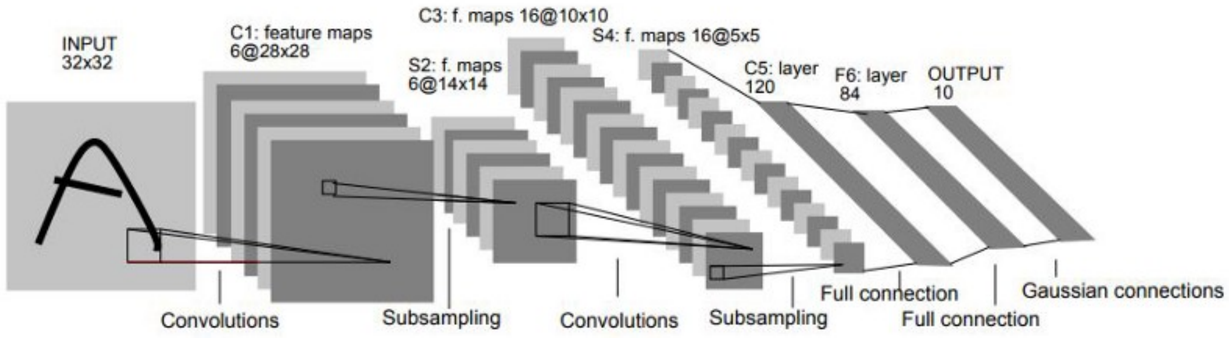
In the Previous Episode

- Point cloud processing
  - Symmetries in deep learning
  - PointNet and it’s successors
- Today: polygonal meshes, going beyond point clouds

Data Symmetries

In general

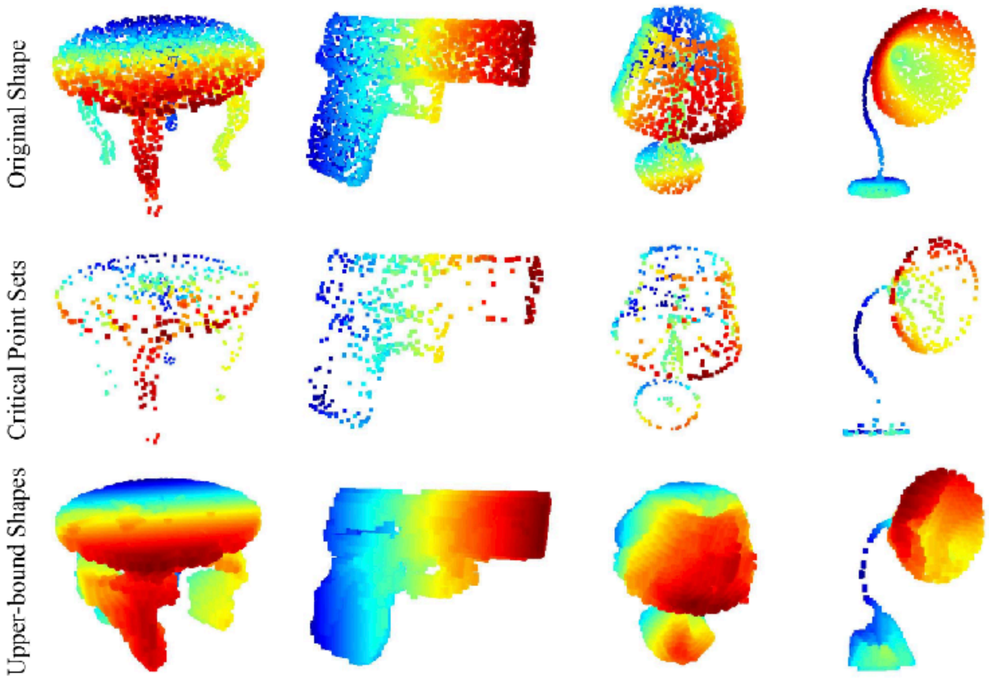
- Consider function f and a group G acting on $\mathrm{dom}\, f$
- **Equivariance:** $\forall g \in G \;\; f(g \cdot x) = g \cdot f(x)$
- **Invariance:** $\forall g \in G \;\; f(g \cdot x) = f(g)$
- Deep architectures exploit symmetries by stacking equivariant layers



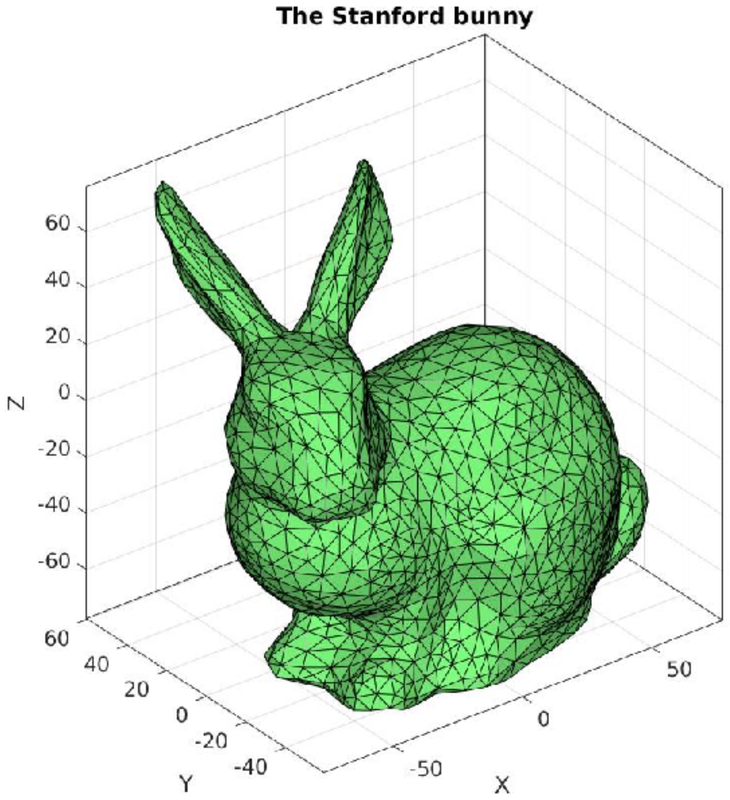
Gradient-based learning applied to document recognition Yann Lecun, Léon Bottou, Yoshua Bengio, Patrick Haffner

PointNet

- Permutation invariance for learning on sets
  - $f(x_1, \ldots, x_n) = g\left(\mathrm{MAX}(h(x_1), \ldots h(x_n))\right)$
  - In practice, h and g are MLPs
- Rotation invariance with Vector Neurons
- Hierarchical features & Transformers



Today: Meshes & (Differentiable) Rendering

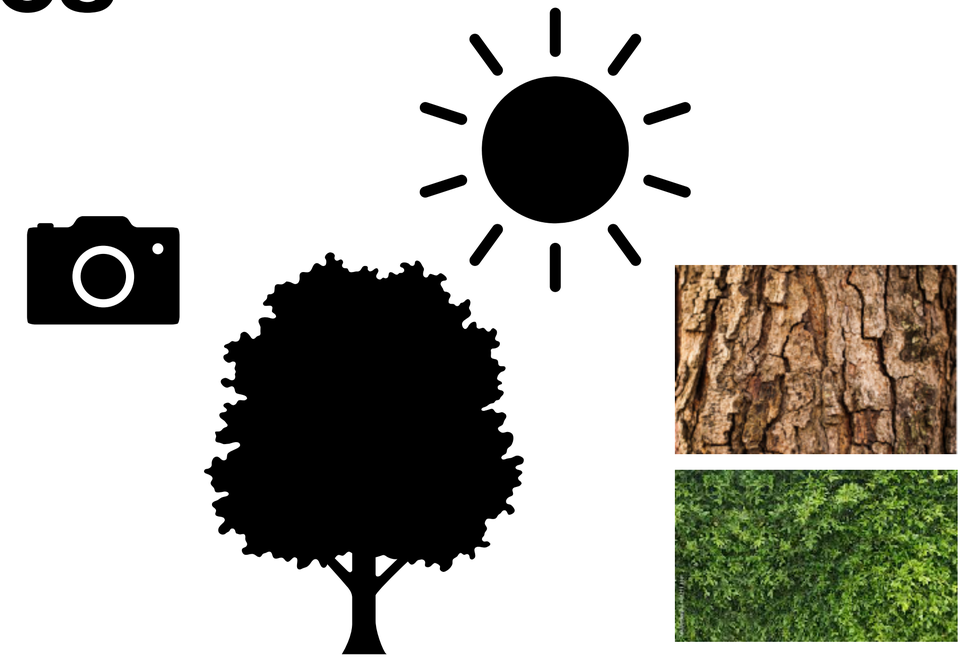


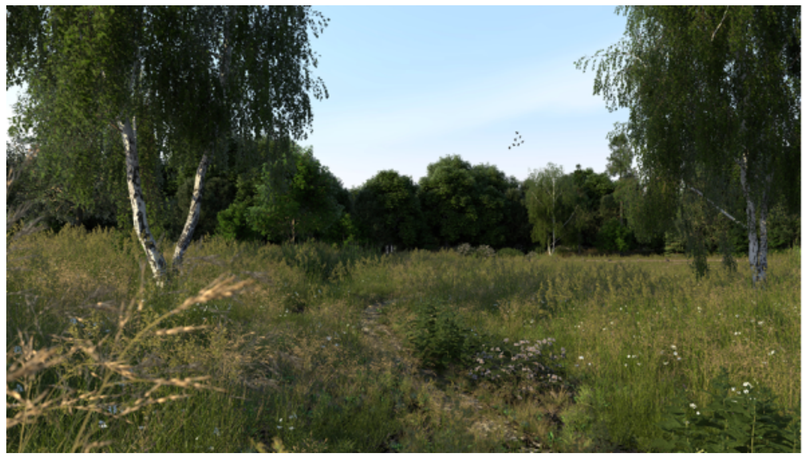
https://en.wikipedia.org/wiki/Stanford_bunny
http://l2program.co.uk/786/little-tweaks-here-and-there

Meshes & 3D Scenes

Modern Computer Graphics

- Scenes
  - Camera
  - Geometry
  - Textures
  - Light
- Renderer
  - Produces an image of a scene

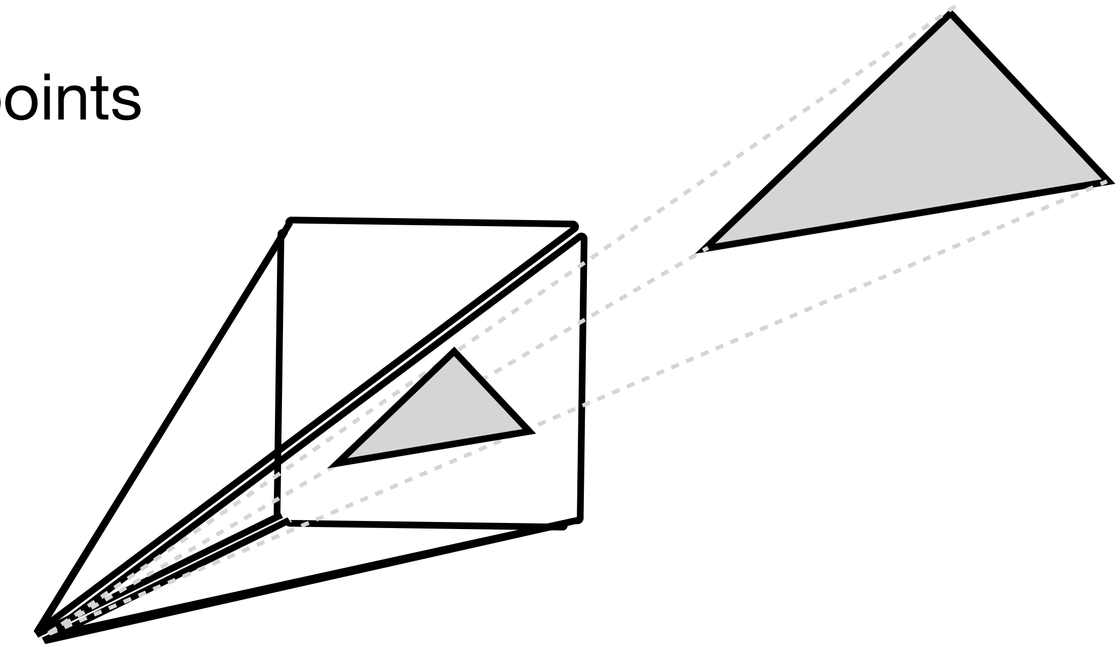




https://pbrt.org/gallery

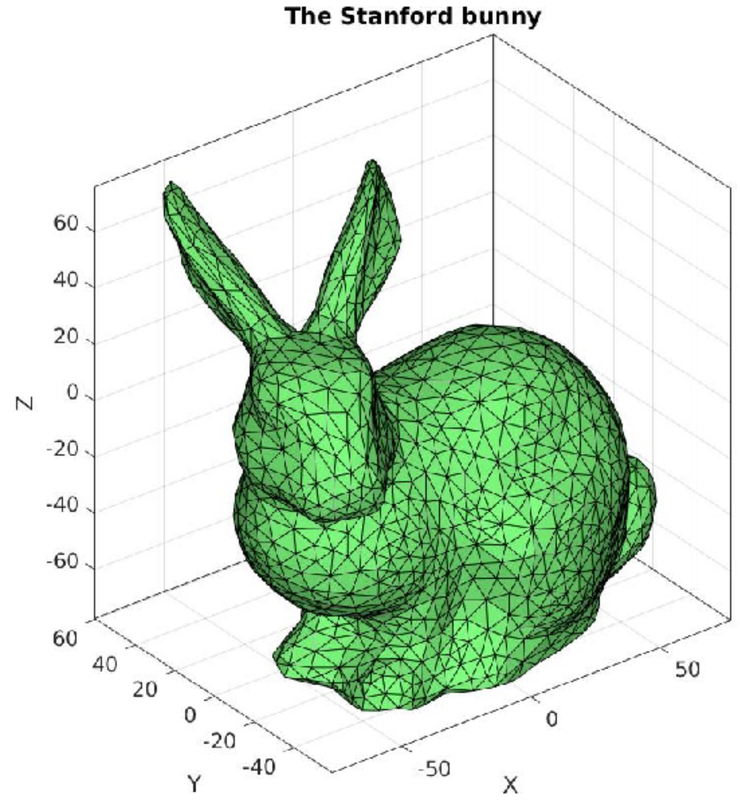
Triangles as a Primitive

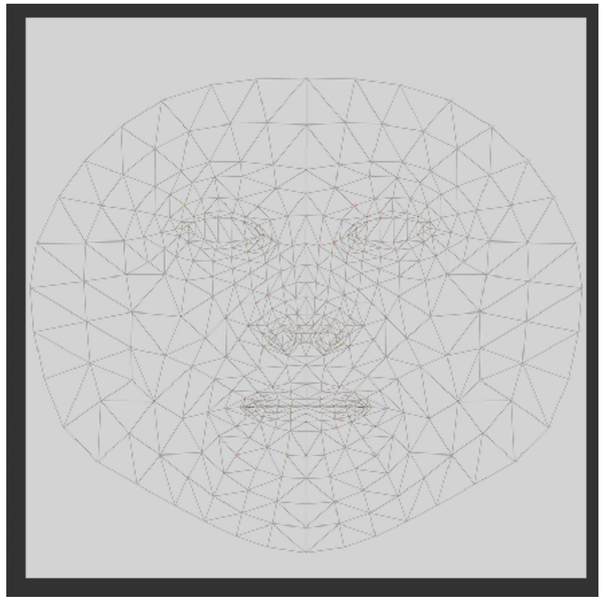
- Multiple ways to represent shapes
  - Splines, level sets, polygonal meshes, points
- Triangles are very convenient
  - Three points always lie on a plane
  - Still a triangle after projection
  - Easy to tell if a point is inside



Polygonal Meshes
Along with Textures

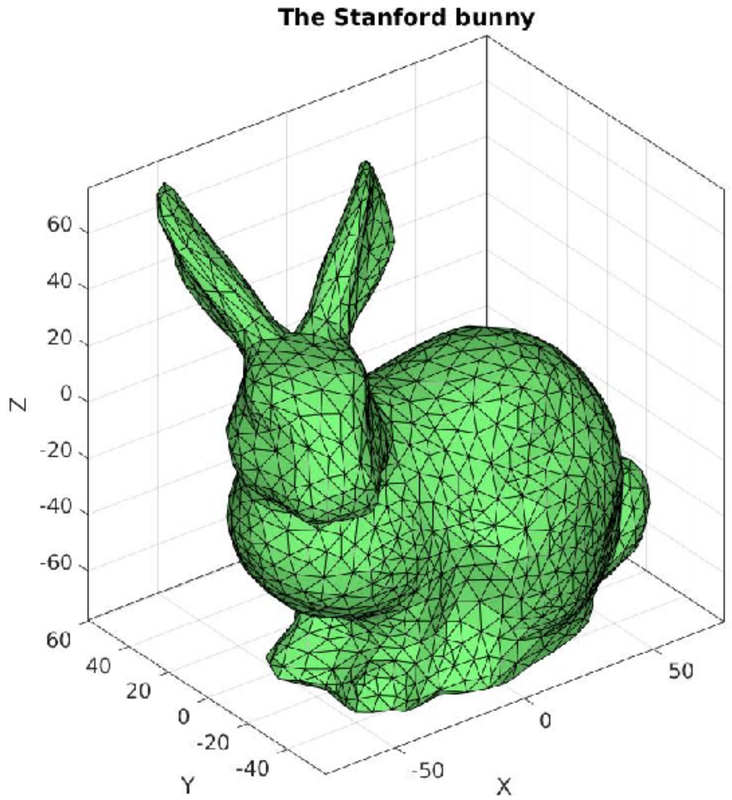
- Mesh consists of
  - Vertices $x_i \in \mathbb{R}^3,\ i = 1,\ldots,N$
  - Triangles $(p_{j0}, p_{j1}, p_{j2}),\ p_{ij} \in \{1,\ldots,N\}$
- Texture
  - Function $T : [0,1]^2 \to \mathbb{R}^C$
  - UV-map: $f : \{1,\ldots,N\} \to [0,1]^2$





https://github.com/google-ai-edge/mediapipe/issues/1698

Physically Based Rendering

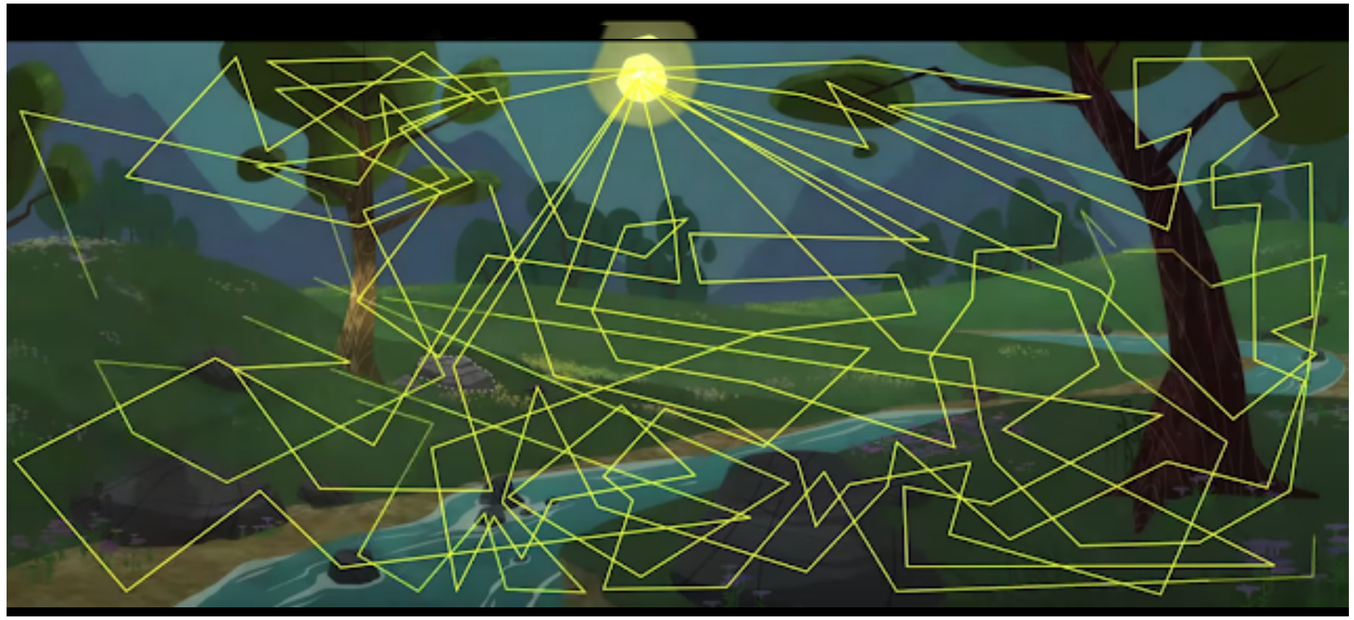


graphics

vision

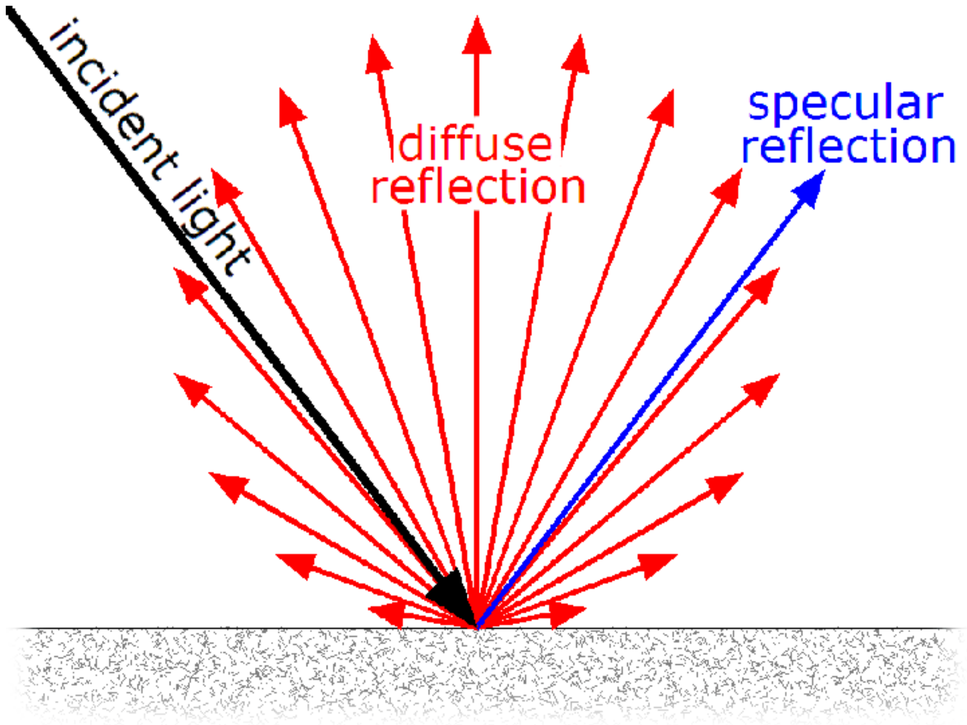
Path Tracing

- Idea: physical simulation of light propagation
- Emit *multiple* rays
- Rays interact with scene geometry
- Register rays that reach camera sensor



Light Interaction

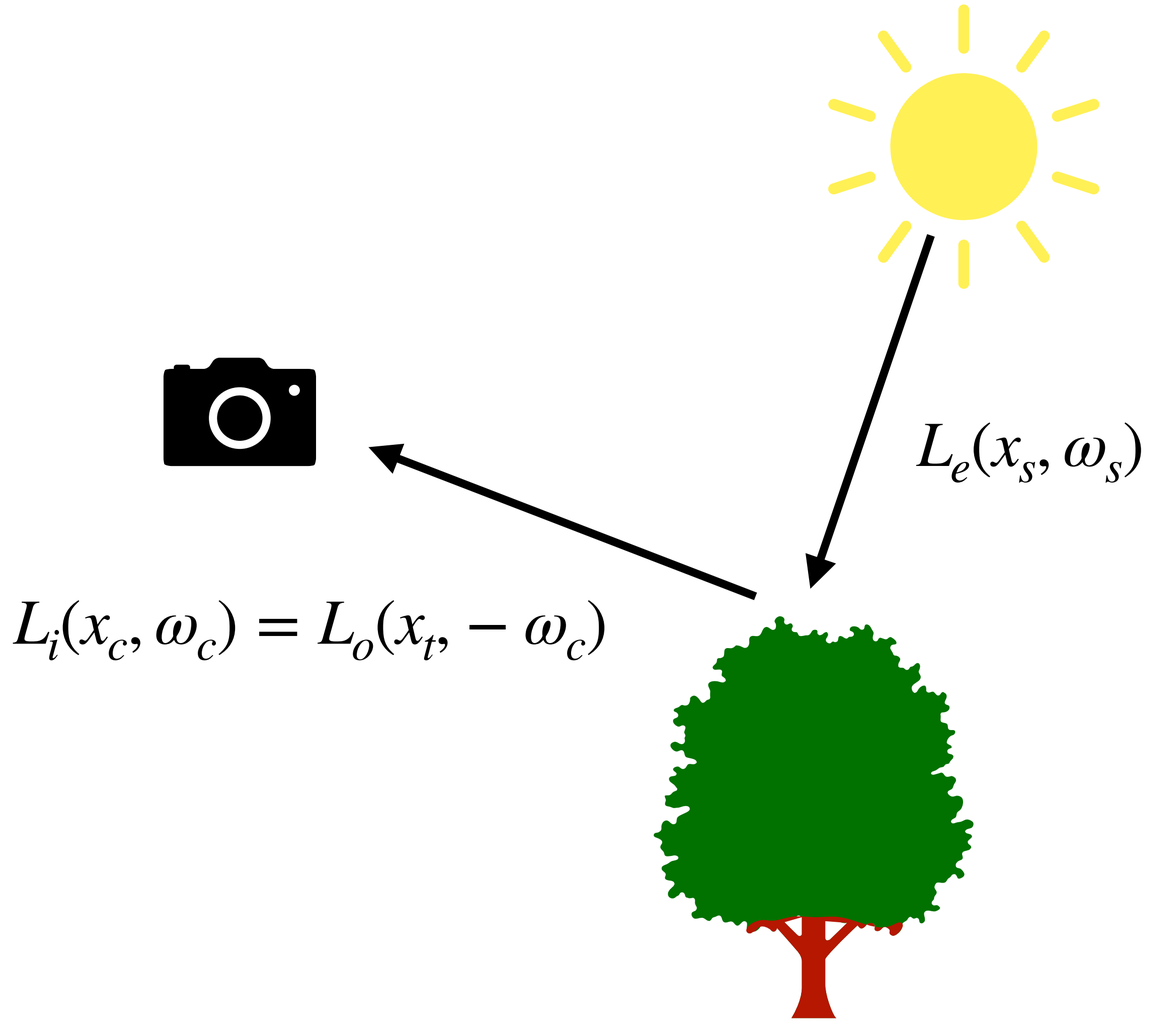
- Diffuse reflection
  - All reflection directions are equal
- Specular reflection
  - Perfect mirror
- In general: BRDF
  - Bidirectional Reflectance **Distribution** Function
  - Function $f_r(\omega_i, \omega_r) : [0,1]^2 \times [0,1]^2 \rightarrow \mathbb{R}$



Radiance Functions

Aggregating Simulation Results

- Three kinds of radiance
  - Emitted $L_e(x, \omega_o)$
  - Outgoing $L_o(x, \omega_o)$
  - Incoming $L_i(x, w_i)$
- Rendering amounts to computing $L_i(x, \omega)$
  - Point x being the camera location
  - Angle grid $\omega \in \Omega$ covering the virtual image plane



Rendering Equation

Framework Behind Path Tracing

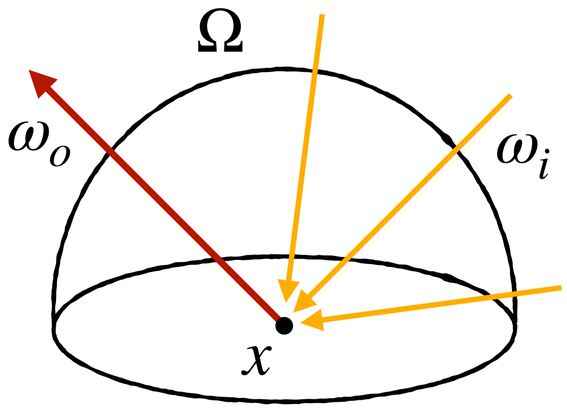
emitted radiance in ω_o

incoming radiance

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i$$

outgoing radiance in ω_o

angle weight of photon density



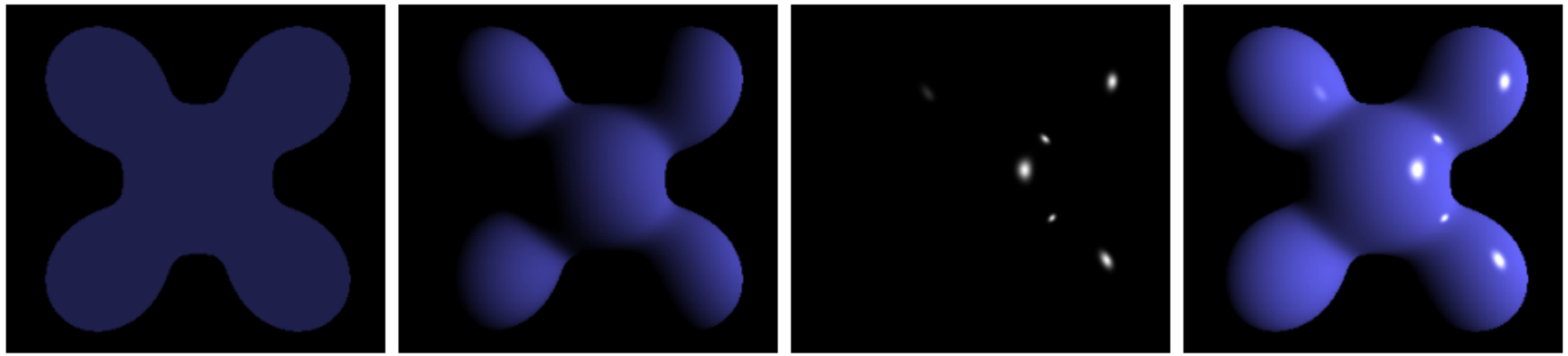
fraction of light that is reflected into ω_o

Kajiya J. T. The rendering equation // 1986

Side Note on PBR Textures

Two Kinds of Reflections

- Ambient texture $T_a \in \mathbb{R}^{H \times W \times 3}$
- Diffuse texture $T_d \in \mathbb{R}^{H \times W \times 3}$
- Specular Texture (metallic & roughness) $T_s \in \mathbb{R}^{H \times W \times 2}$



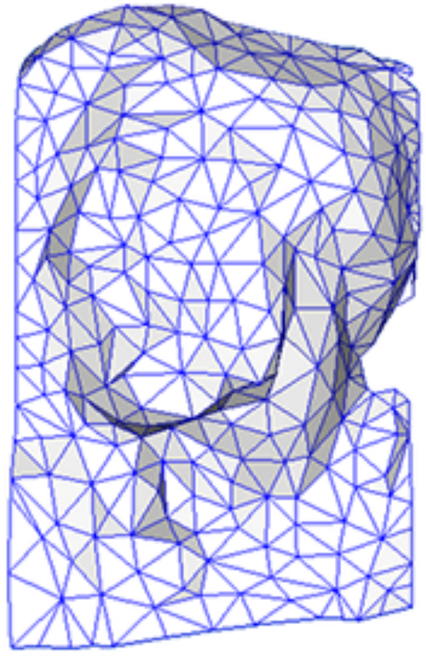
https://en.wikipedia.org/wiki/Phong_shading

Side Note on PBR Textures

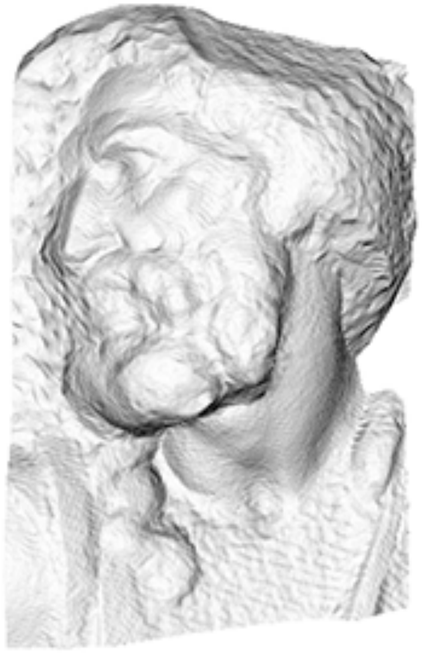
Normal Maps

- Normal map $T_n \in \mathbb{R}^{H \times W \times 2}$

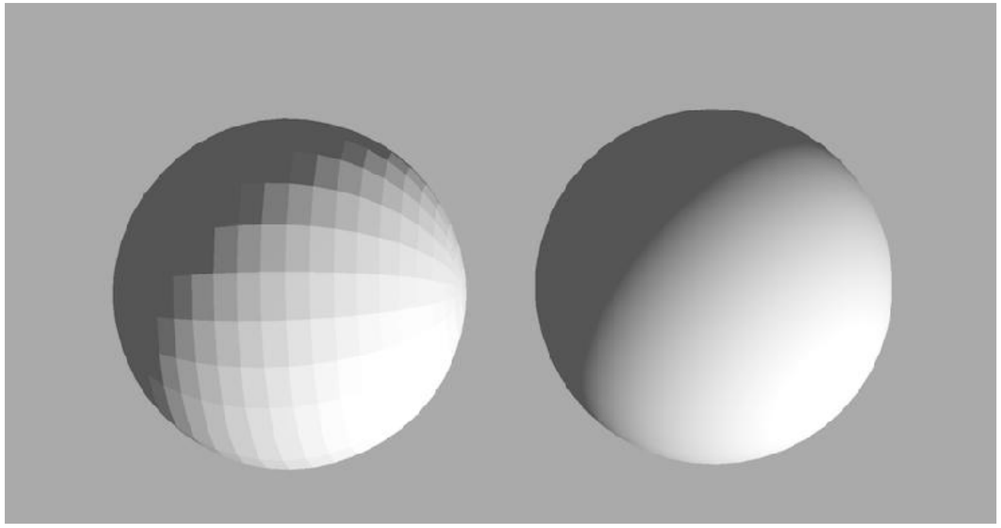
original mesh
4M triangles



simplified mesh
500 triangles



simplified mesh
and normal mapping
500 triangles



https://learnopengl.com/Advanced-Lighting/Normal-Mapping

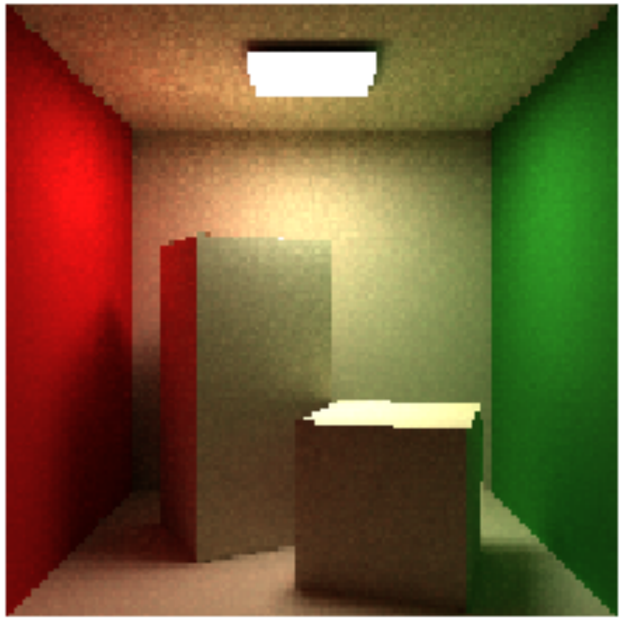
Solving Rendering Equation

Naive Rendering Pseudo-Code

```
for each pixel (x, y)
   compute ray origin r_o and direction r_d
   for each object O in scene
      if ray intersect O and is closest for far
         save O_closest
   compute emitted radiance at intersection with O_closest
   recursively compute incoming radiance at the intersection
```

Implementations

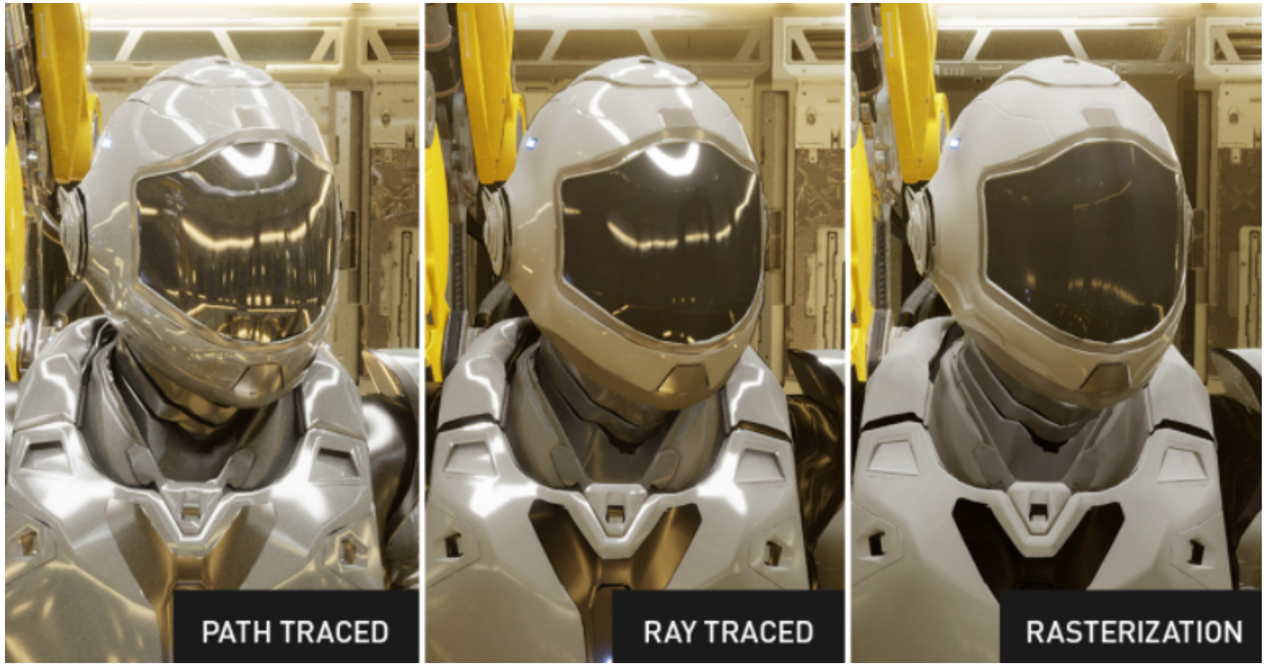
- Simple implementation in JAX & more detailed explanation
  - https://blog.evjang.com/2019/11/jaxpt.html
- Mitsuba Renderer
  - https://www.mitsuba-renderer.org/
- Physically-based rendering in 3D reconstruction
  - https://nvlabs.github.io/nvdiffrecmc/



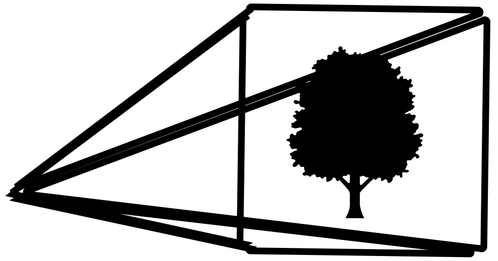
Rasterization

Isn’t Path Tracing Enough?

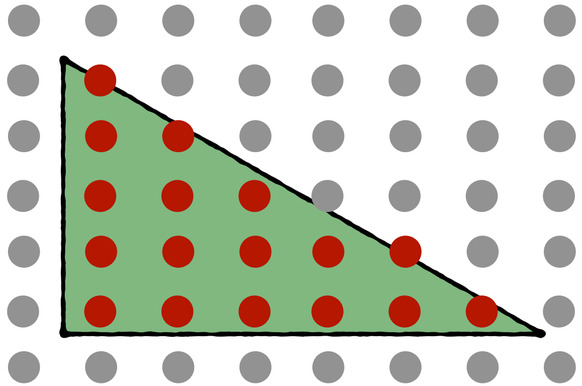
- Path tracing is slow and noisy
  - Predominantly used in cartoons & movies
  - Real-time rendering is still an ongoing effort
- Rasterization is an alternative rendering approach
  - Hard to make photorealistic
  - The main approach behind real-time graphics

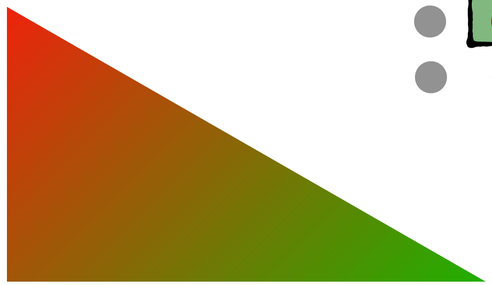


Rasterisation Pipeline

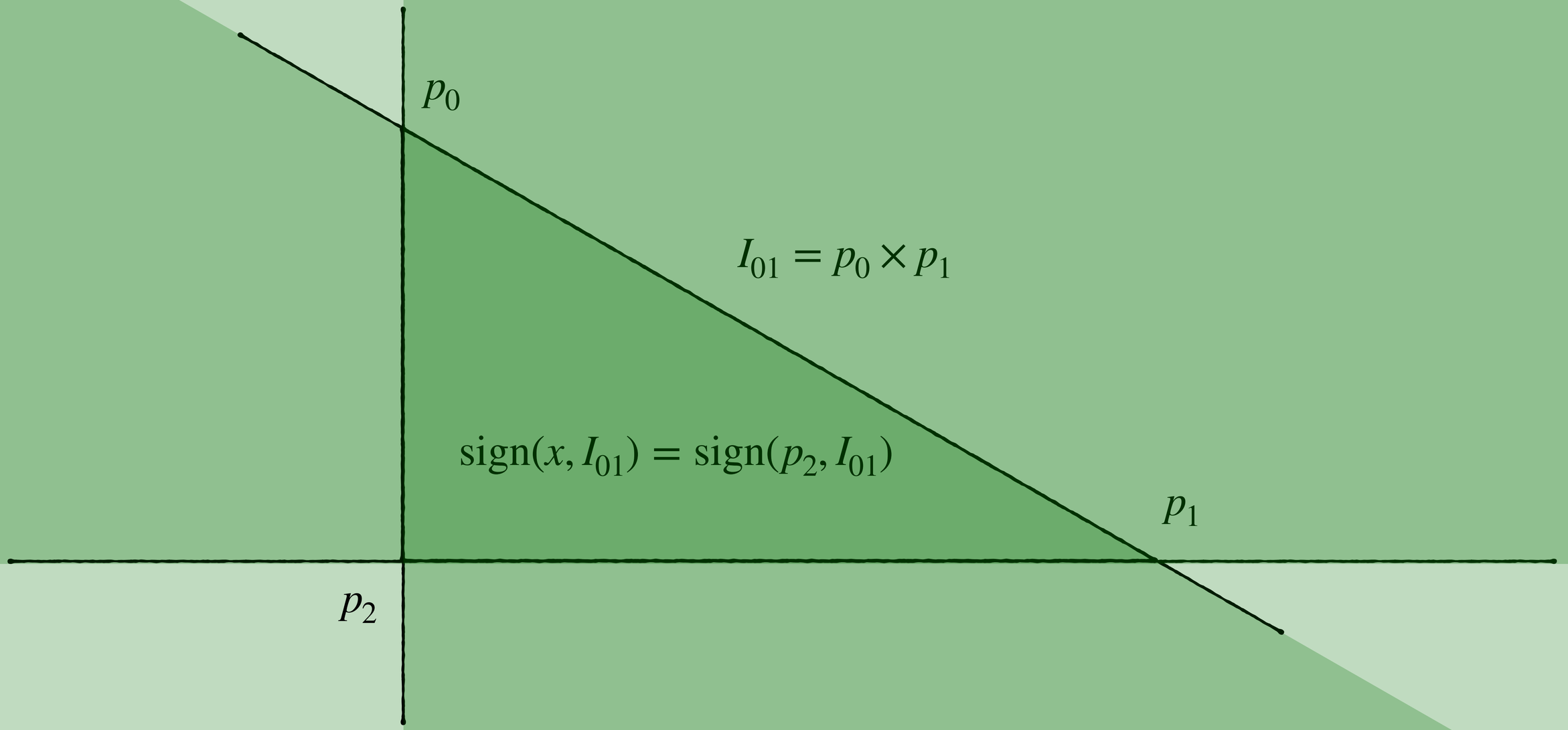


- Position objects in the world C^{w2c}
- Project objects onto the screen
- For each triangle
  - Sample coverage
  - Interpolate triangle attributes
  - Sample textures / evaluate shaders
  - Combine into the final image





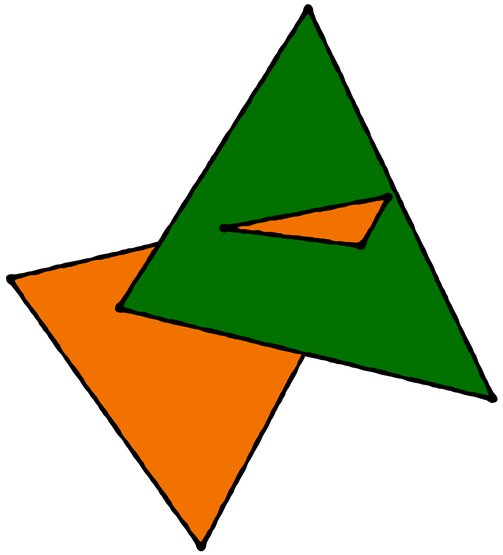
Drawing a Single Triangle

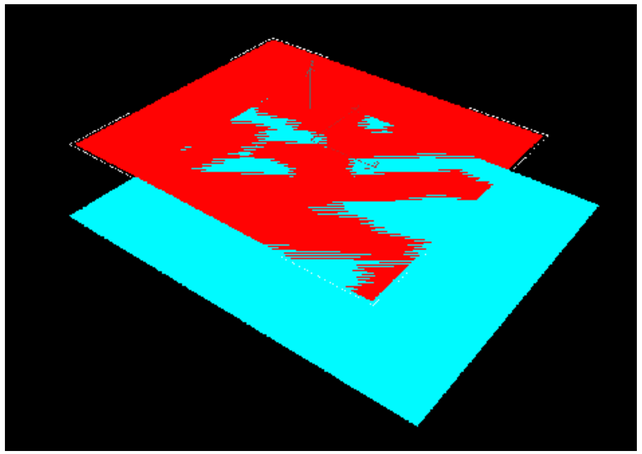


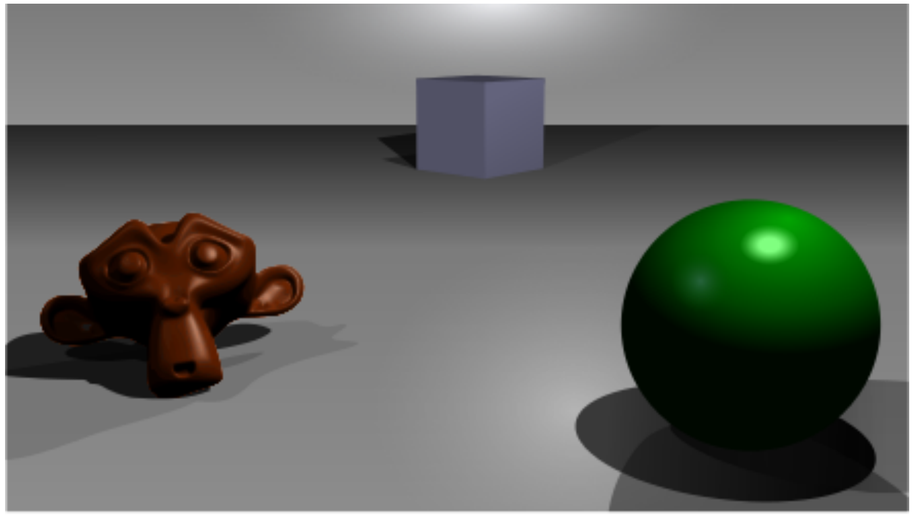
Drawing Two Triangles

Depth Buffer

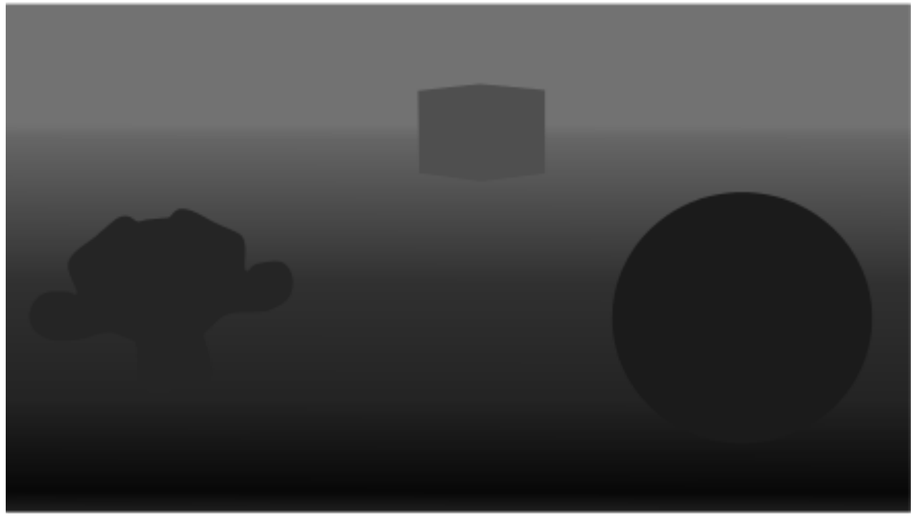
- Two triangles may intersect
- Need to render only the closest points
- Solution: store depth buffer







A simple three-dimensional scene



Z-buffer representation

https://en.wikipedia.org/wiki/Z-buffering https://en.wikipedia.org/wiki/Z-fighting

Extended Projection Matrix

- Projection matrices we encountered omitted one coordinate

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- The omitted coordinate stores depth
- Modify matrix to store depth

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_n + z_f}{z_n - z_f} & \frac{2 z_f z_n}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

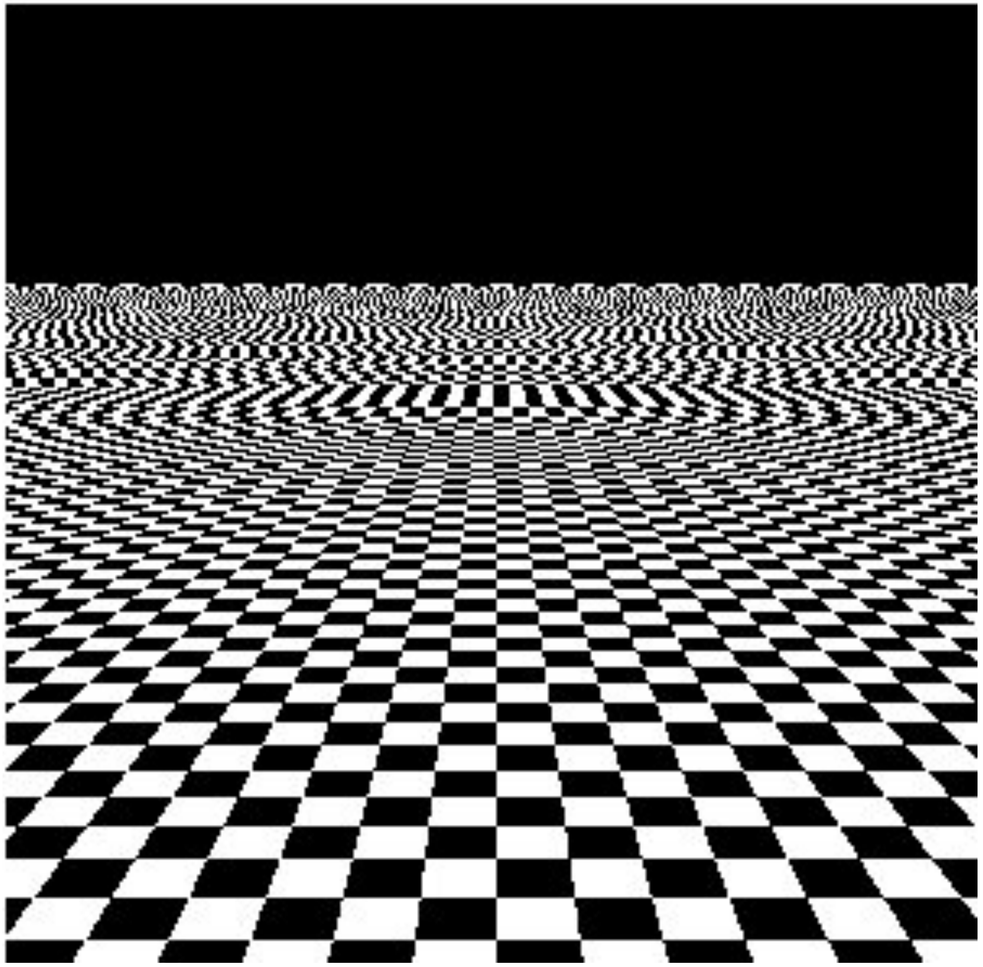
- Maps depth $[z_n, z_f]$ into $[-1,1]$

Pseudo-Code for Rasterisation

```
for each triangle O in scene  # from closest to farthest
   project and transform O onto the screen
   for each pixel (x, y)
      if (x, y) is in O and (x, y) is closest so far
         compute color at (x, y)
         save corresponding depth to depth buffer
```

Aliasing

One of the Many Details Swept Under the Carpet



https://textureingraphics.wordpress.com/what-is-texture-mapping/anti-aliasing-problem-and-mipmapping/

Colour of a Point

pixel #1

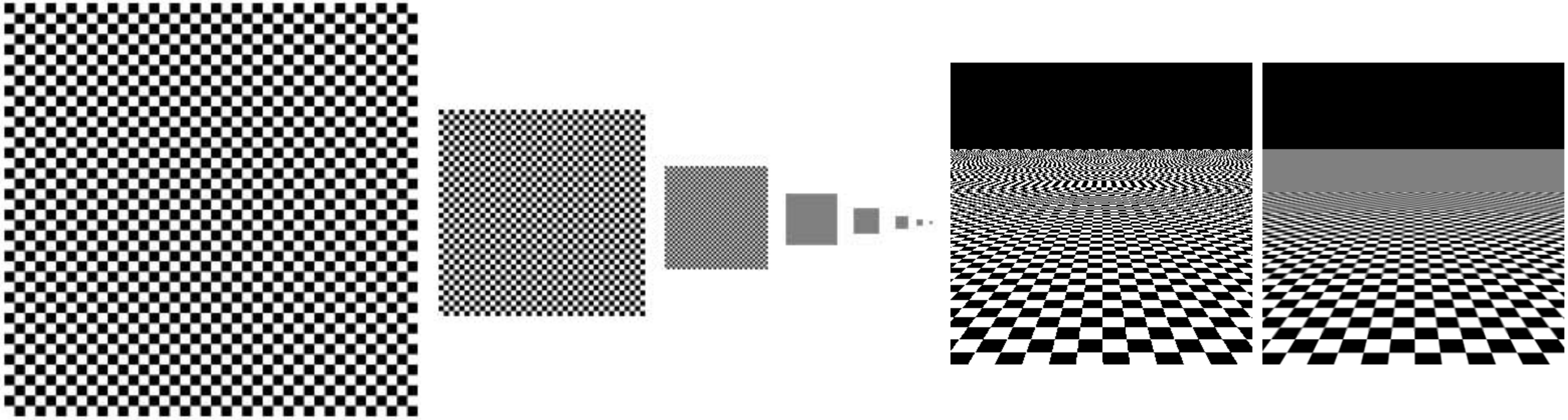
pixel #2

pixel #3

- Ideally, the resulting color should be the average color across the pixel

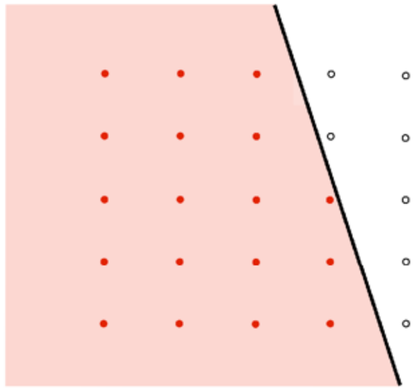
MIP-Mapping

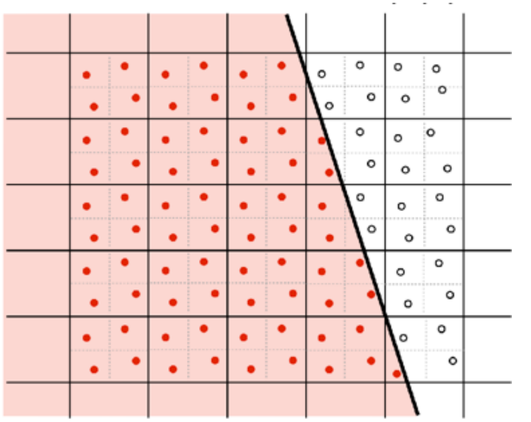
- Computing the average for each pixel would be too expensive
- Solution: pre-compute texture at different scales

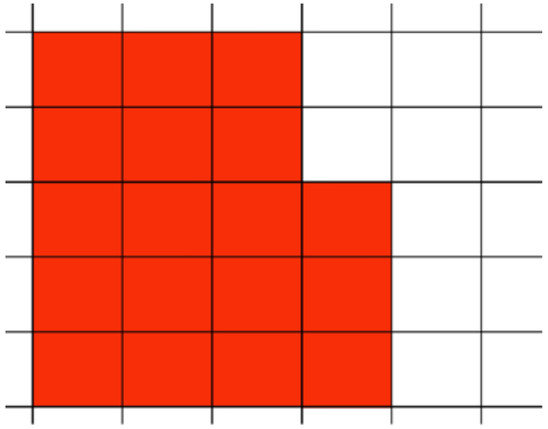


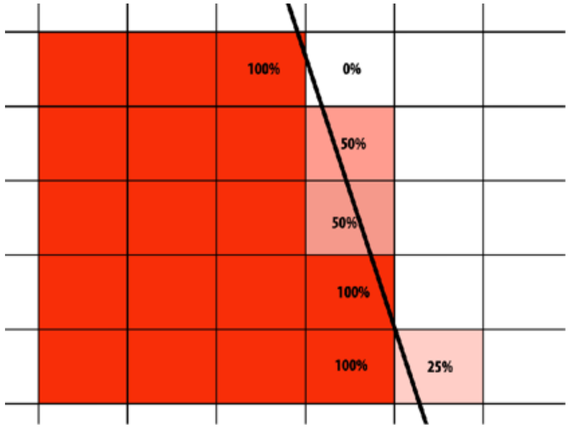
https://textureingraphics.wordpress.com/what-is-texture-mapping/anti-aliasing-problem-and-mipmapping/

Border Aliasing



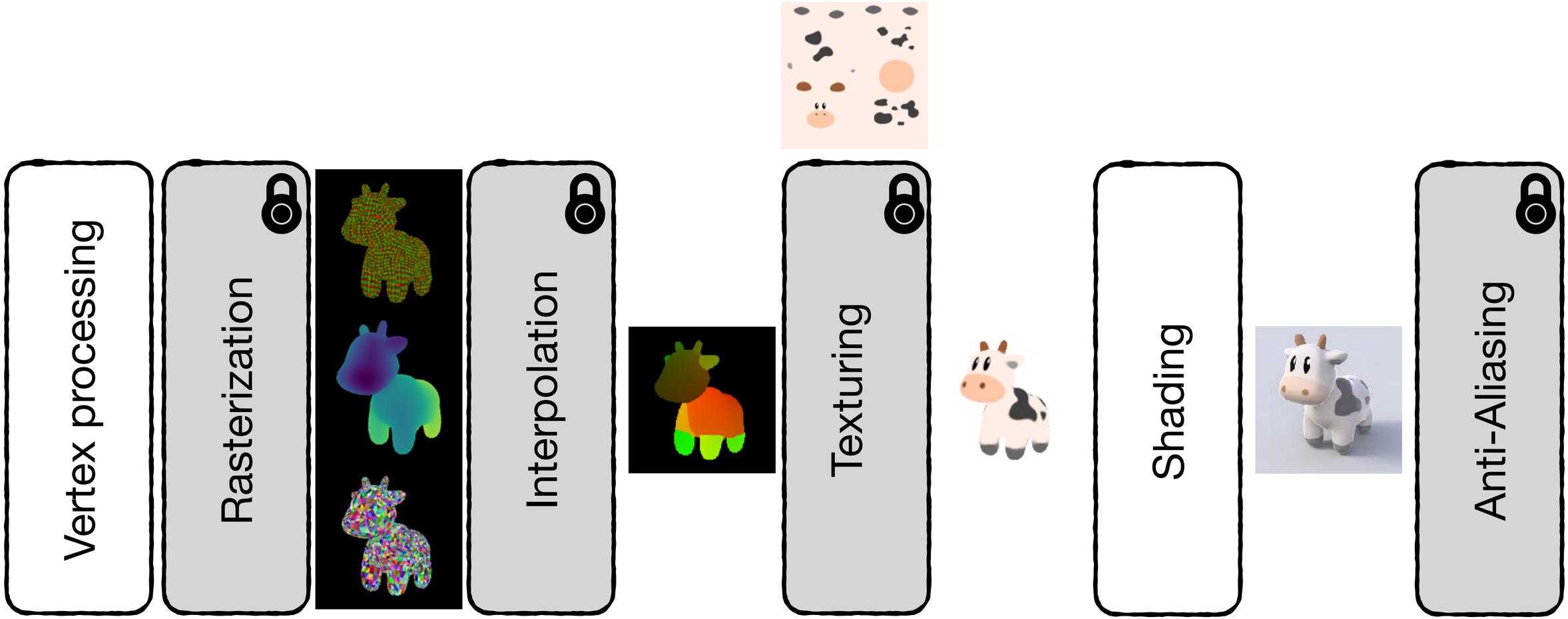






Images taken from CMU Graphics Course https://www.youtube.com/watch?v=B0hsT2nplc0

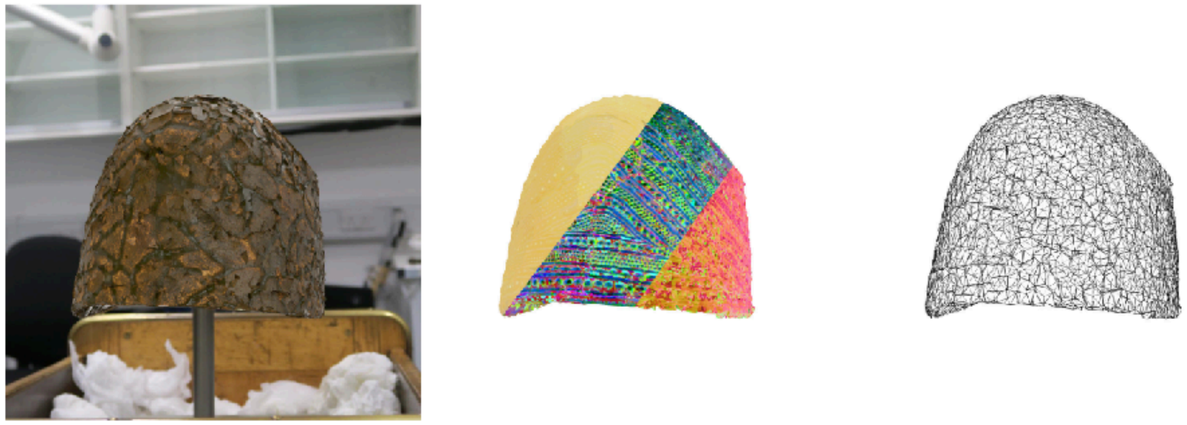
nvdiffrast



https://nvlabs.github.io/nvdiffrast/
Laine S. et al. Modular primitives for high-performance differentiable rendering // 2020
Slide adapted from presentation on differentiable rendering by Dmitry Senyushkin & Nikolay Patakin,

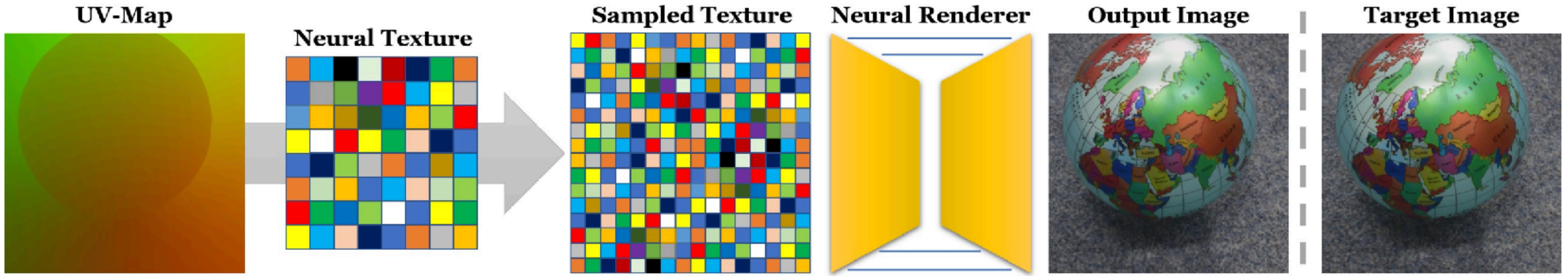
Applications: nvdiffrec

Reference photo | Scene editing | Cloth simulation

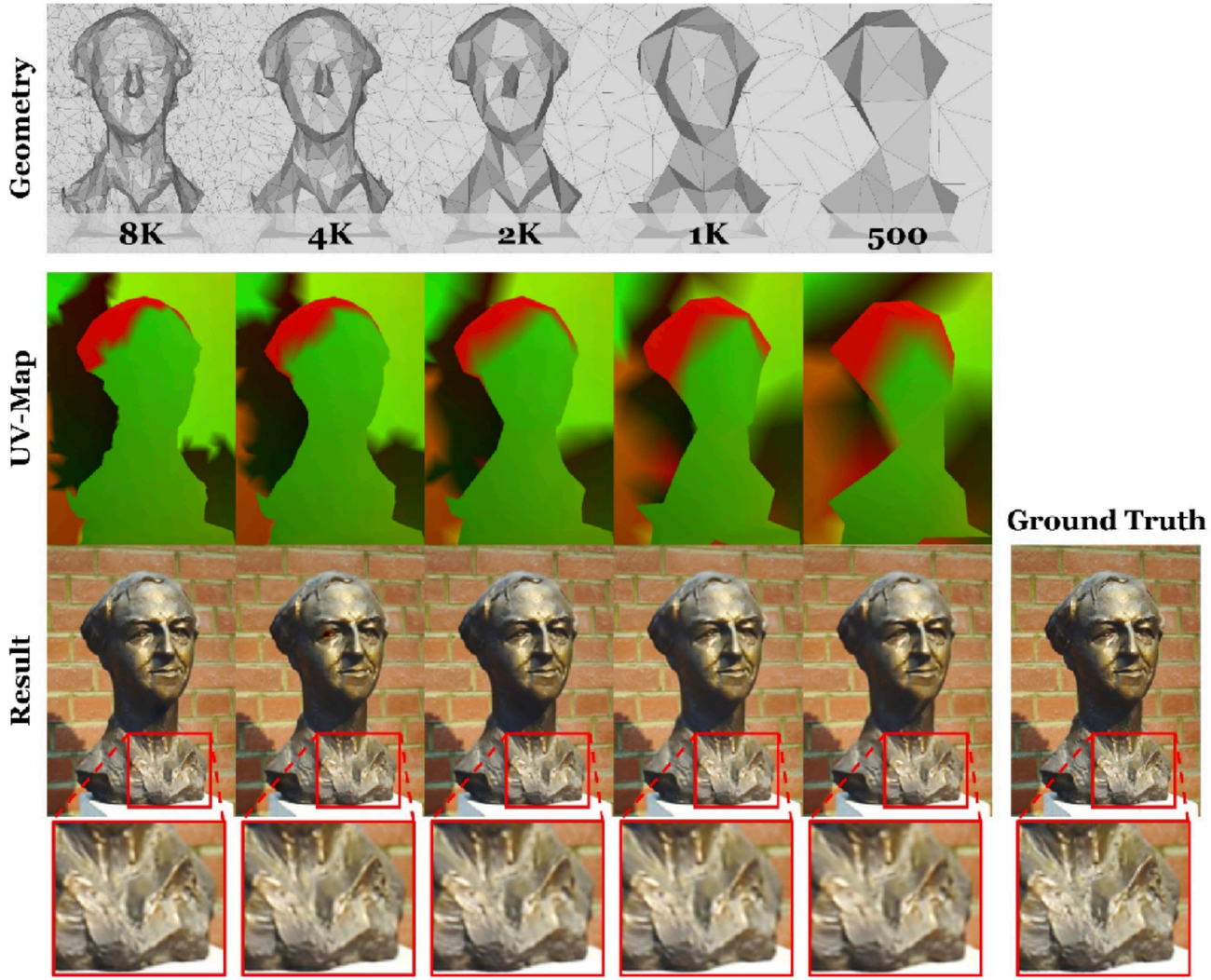


Material editing | $\boldsymbol{k}_d/\boldsymbol{k}_{\text{orm}}/\mathbf{n}$ | Mesh

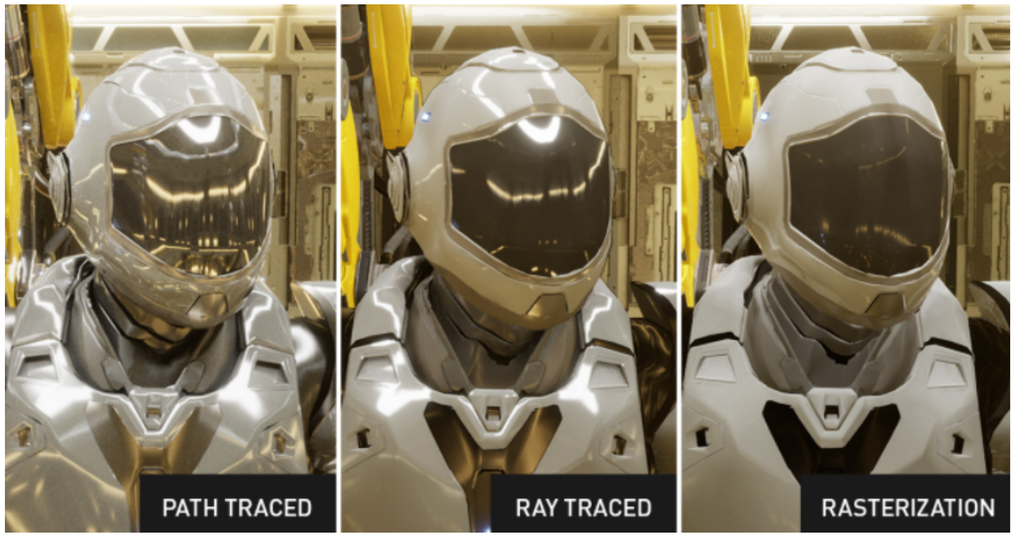
Applications: Deferred Neural Rendering



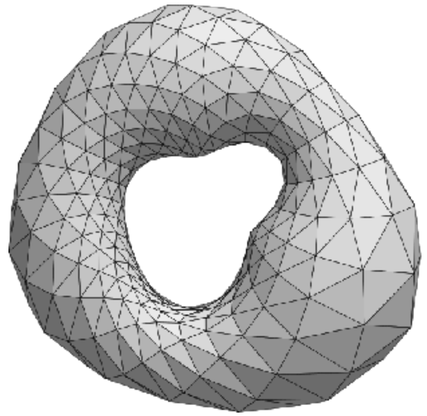
Applications: Deferred Neural Rendering

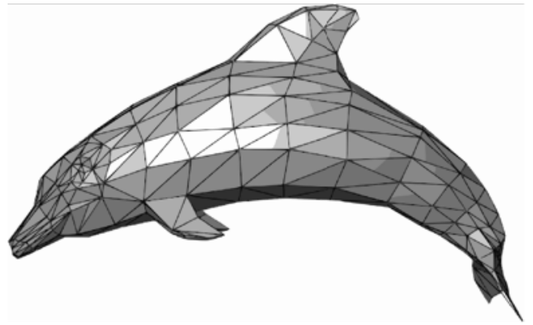


Key Takeaways



- Path tracing
  - Photorealistic & Elegant
  - Slow
- Rasterisation
  - Hard to make realistic renders
  - Fast





- Both allow differentiation for inverse graphics, rasterisation is a little more common
- Next time: implicit representations for 3D data

https://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/