CS148 Summer 2025 - Lecture 8

07/17/2025

Sampling and Textures

cs148.stanford.edu

Guest Lecture Next Thursday

Catherine Hicks

Catherine has a BFA in character animation and has been in animation ever since. She started in puppet design for sesame street and then moved into animation at Pixar in 2009, working on films, including Toy Story 3, Coco, Inside Out and many more. She became an animating director in 2022, and in 2024 moved to Cartwheel, an AI-enabled animation tool system, as the head of animation innovation.

Lecture Outline

- Announcement: Project Proposal due next Thurs with HW4
  - Proposal is extra credit and makes up for 5 missed points across HW & Quiz grades
  - A reference image + 1-2 paragraphs – give us enough info so we can give feedback!

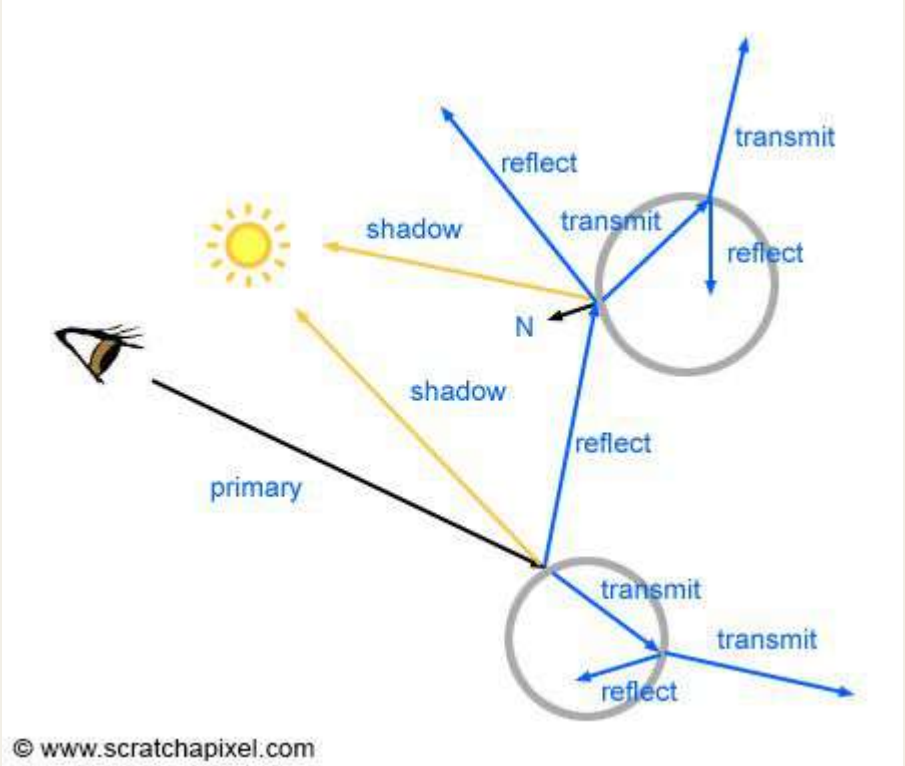
- Wrapping up Ray Tracing:
  - How to deal with Area Lights? Depth of field? Color bleeding?
  - Sampling (e.g. Monte Carlo method) -> Stochastic Ray Tracing
  - Path Tracing for Global Illumination

- Textures:
  - Texture Mapping: UV Coordinates and Creation
  - Texture Lookup: Interpolation and Sampling
  - Texture Acquisition: Art, Capture, Procedural
  - Texture Frameworks: One for each BxDF parameter
  - Normal Mapping: “Hacking” geometry with textures
  - Environment Mapping and Sky Boxes

Raytracing so far – Whitted Raytracer

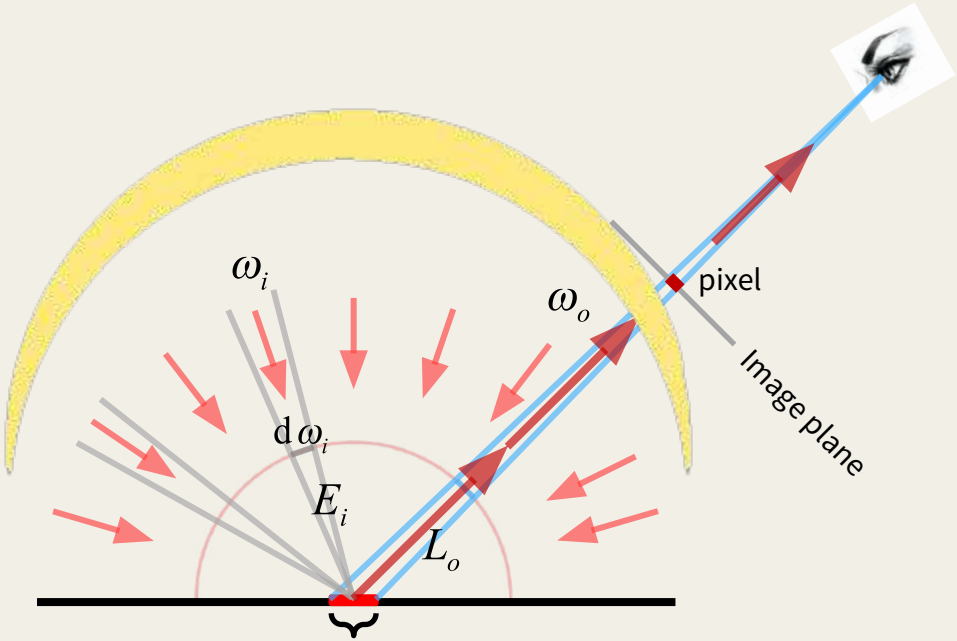
- We've covered the bare bones ray tracing process
- Called a **Whitted raytracer** after Turner Whitted (1980)

- Still begs some questions…
  - area lights?
  - depth of field?
  - color bleeding?



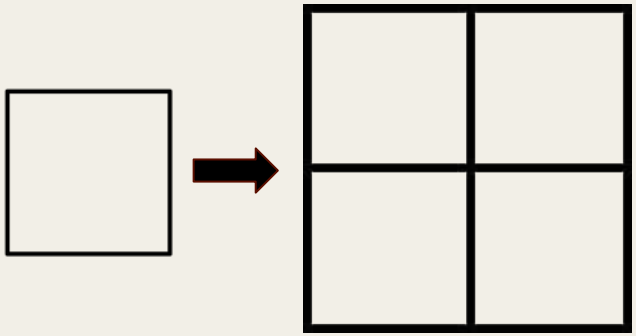
Motivation for Sampling: Area Lights

- Recall an area light can be thought of as a collection of point lights
  Could send out a ray for every point that makes up the area light…

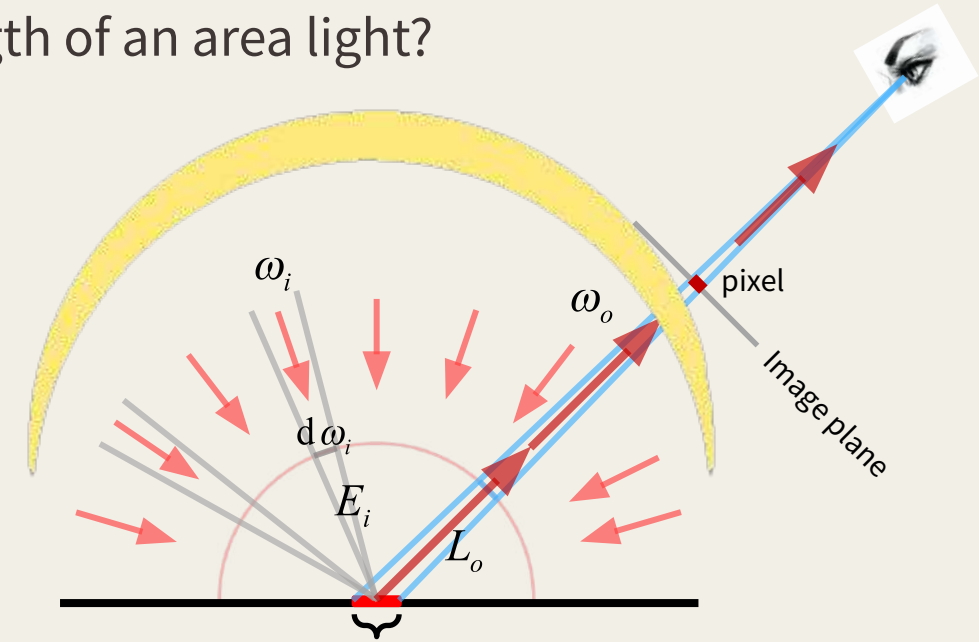


Motivation for Sampling: Area Lights

- Recall an area light can be thought of as a collection of point lights
  Could send out a ray for every point that makes up the area light…
- What if we double the length of an area light?

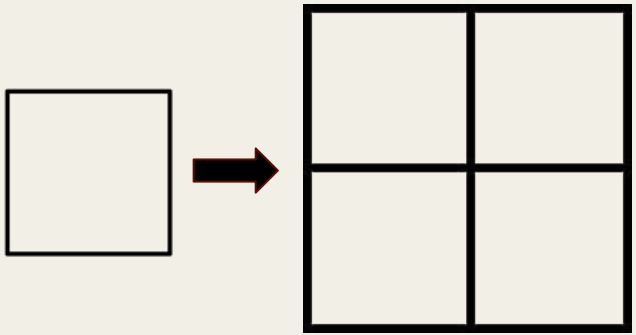


- **4x as many points now!**



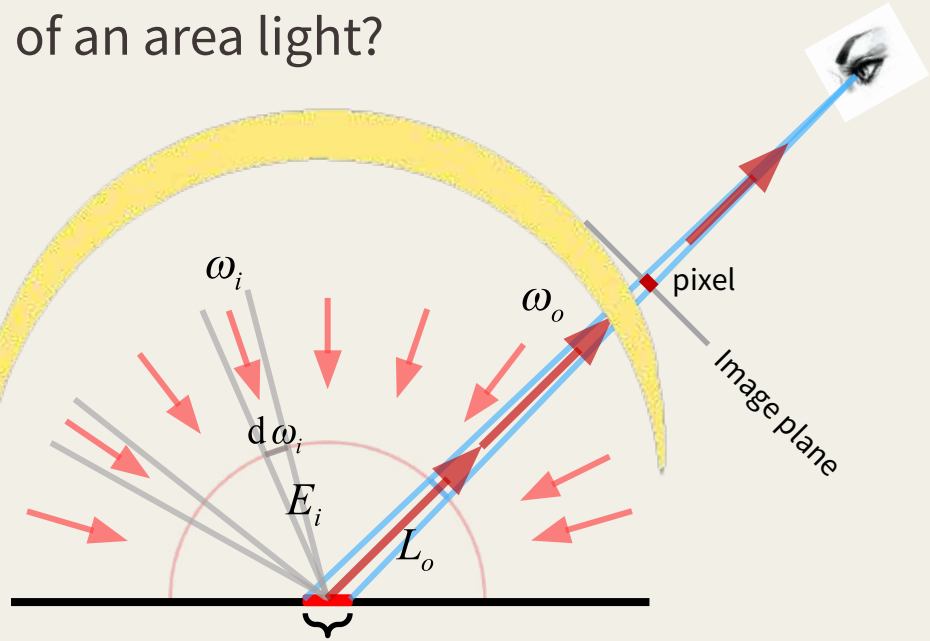
Motivation for Sampling: Area Lights

- Recall an area light can be thought of as a collection of point lights
  Could send out a ray for every point that makes up the area light…
- What if we double the length of an area light?



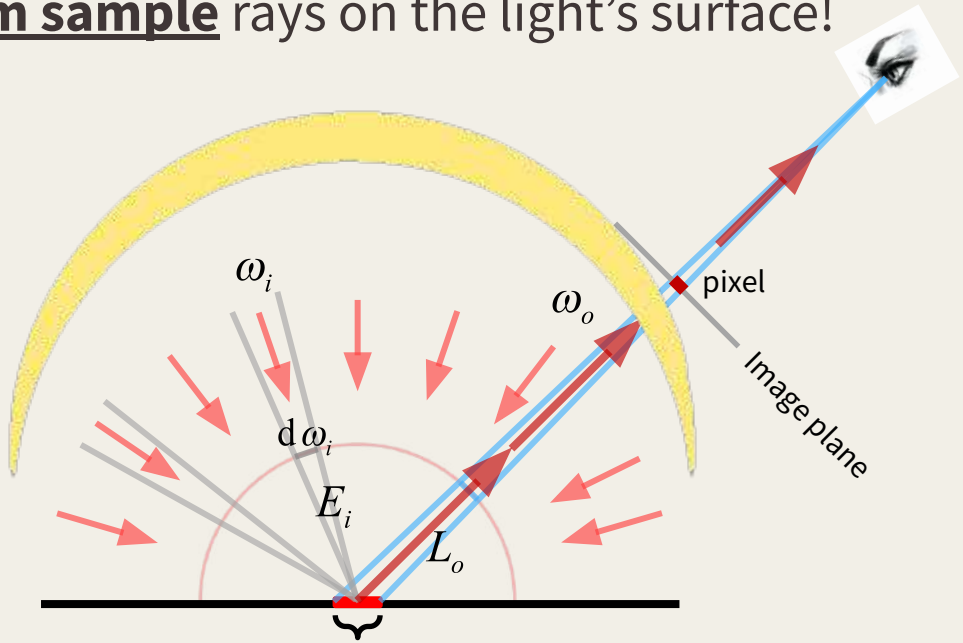
- 4x as many points now!

➡ **Curse of Dimensionality**



Motivation for Sampling: Area Lights

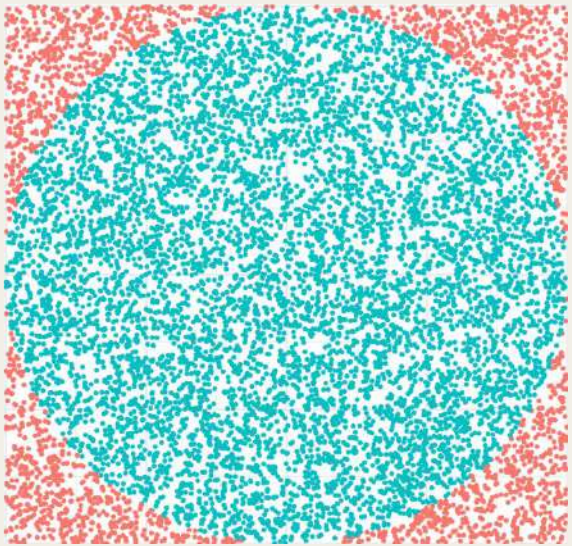
- Recall an area light can be thought of as a collection of point lights
  ~~Could send out a ray for every point that makes up the area light…~~
  Could **random sample** rays on the light's surface!



One Sampling Technique: Monte Carlo

Rely on **repeated random sampling** to get results – example:

- Consider a square of size 2 by 2, then put a circle of radius 1 inside
- Throw darts at the square and color those that hit inside the circle blue, outside red



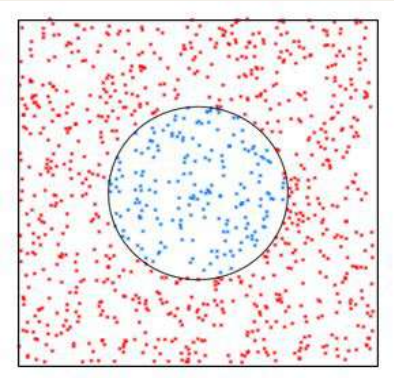
$$\pi \approx 4 \frac{\#blue}{\#red + \#blue}$$

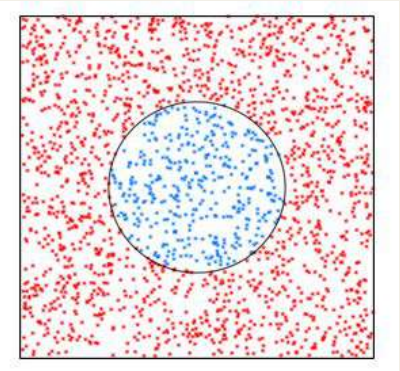
$$\pi \approx 3.1440$$

One Sampling Technique: Monte Carlo

Rely on **repeated random sampling** to get results:

- Slow convergence, but independent of number of dimensions!
- Example: consider a disk-shaped area light
  - randomly sample e.g. 100 points within the area light
  - treat each point as a point light with which to ray trace
  - average the results for the final render!

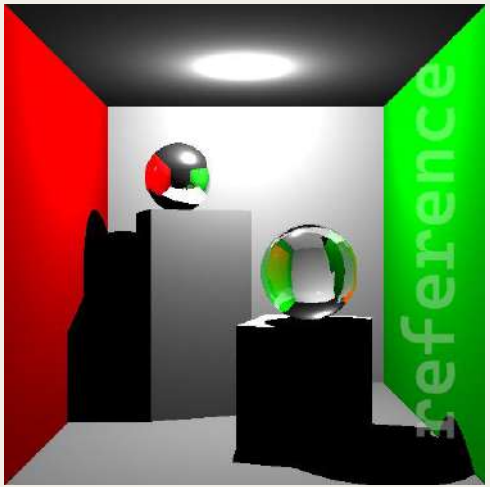




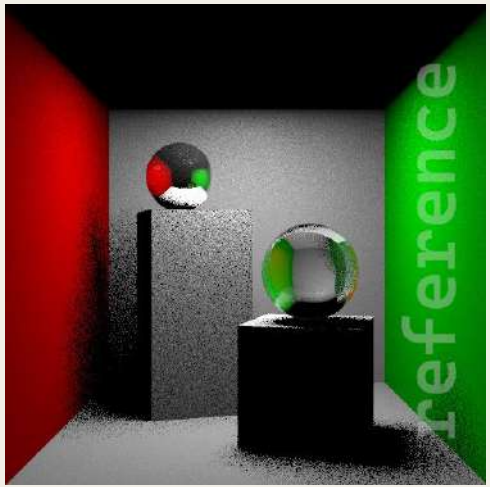
Sampling Area Lights

Rely on **repeated random sampling** to get results:

- randomly sample e.g. 100 points within the area light
- treat each point as a point light with which to ray trace
- average the results for the final render!



ray tracing with just point lights



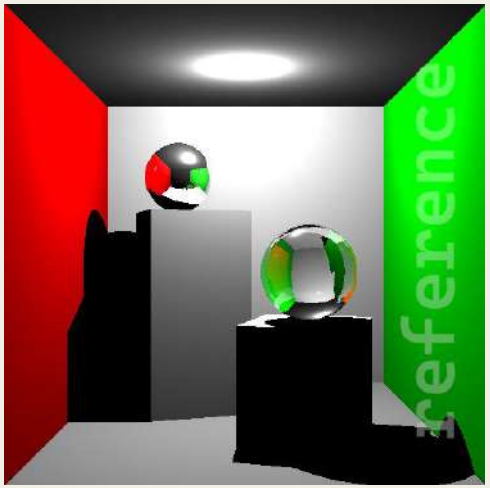
ray tracing with area lights (4 samples)

Sampling Area Lights

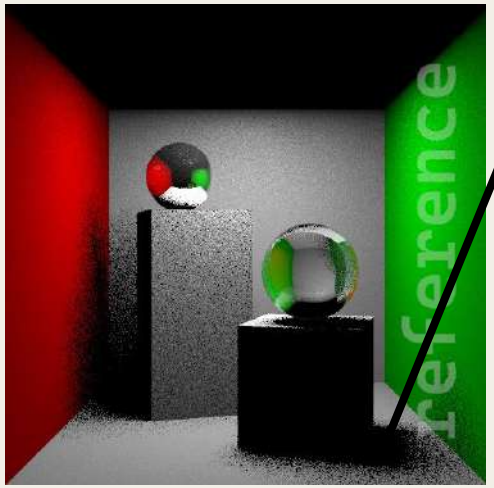
Rely on **repeated random sampling** to get results:

- randomly sample e.g. 100 points within the area light
- treat each point as a point light with which to ray trace
- average the results for the final render!

softer shadows!



ray tracing with just point lights



ray tracing with area lights (4 samples)

Aside: Different Types of Lights

Point Light

Area Light

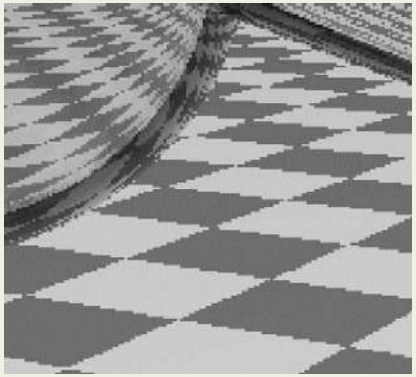
Directional (Sun) Light

Spotlight

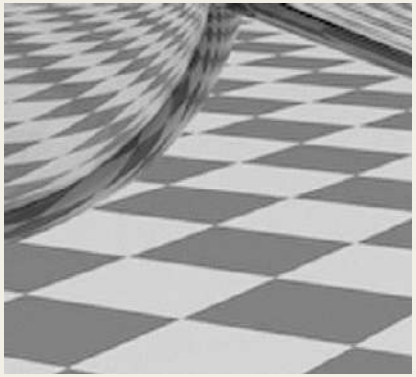
Sampling Camera Rays – Anti-Aliasing

- Simply sending one ray through each pixel of our film plane can lead to “aliasing” or pixelated edges in our render

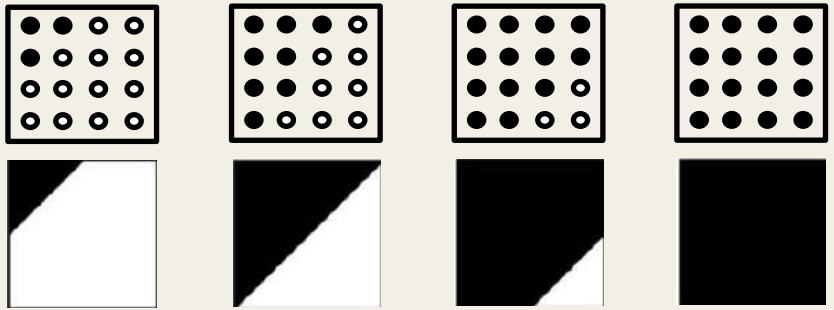
- Make the pixel finer grain and sample camera rays through random points in the pixel, then average the render results!



aliased

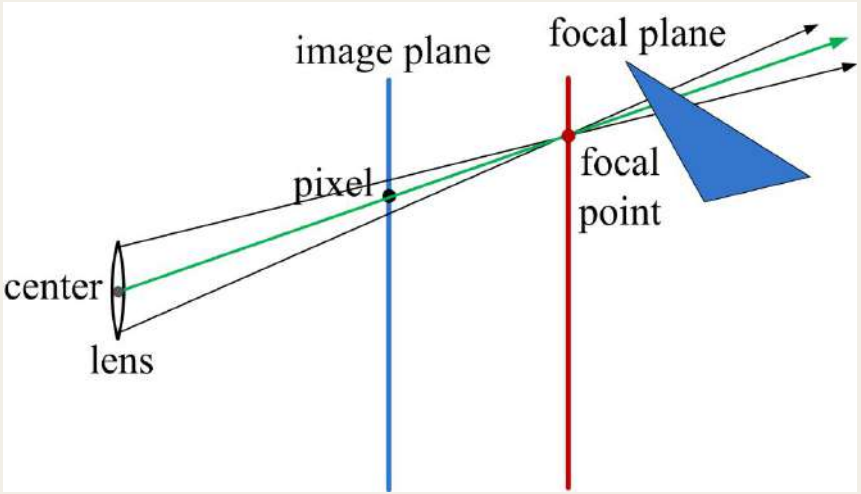


anti-aliased



Sampling Camera Rays – Depth of Field

- Random sample camera rays around the circumference of your lens circle at the camera aperture, then average the result
- Can combine this with anti-aliasing by sending each camera ray through a random part of the pixel as well!



Whitted Raytracer -> Distributed Raytracer

- Also known as stochastic ray tracing
- Use Monte Carlo methods to sample area lights, depth of field, etc to add even more photorealistic detail!

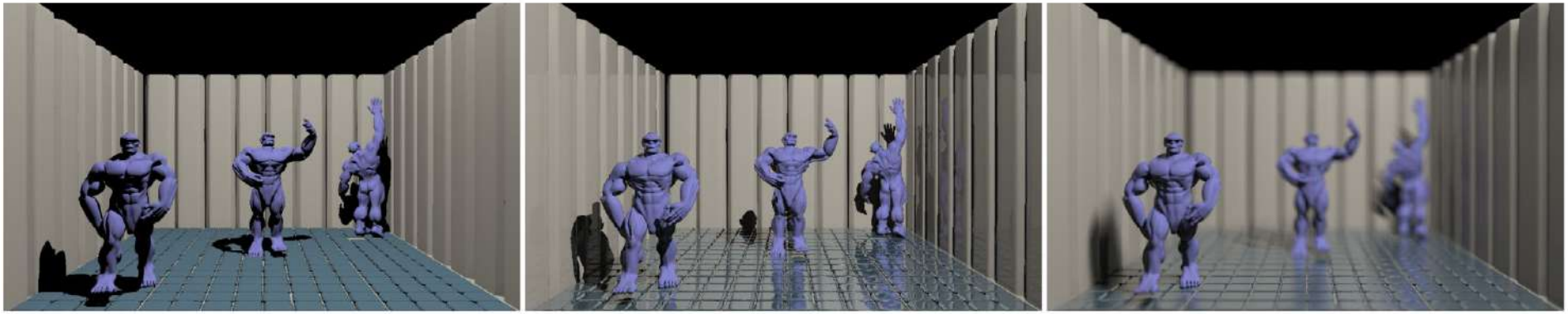
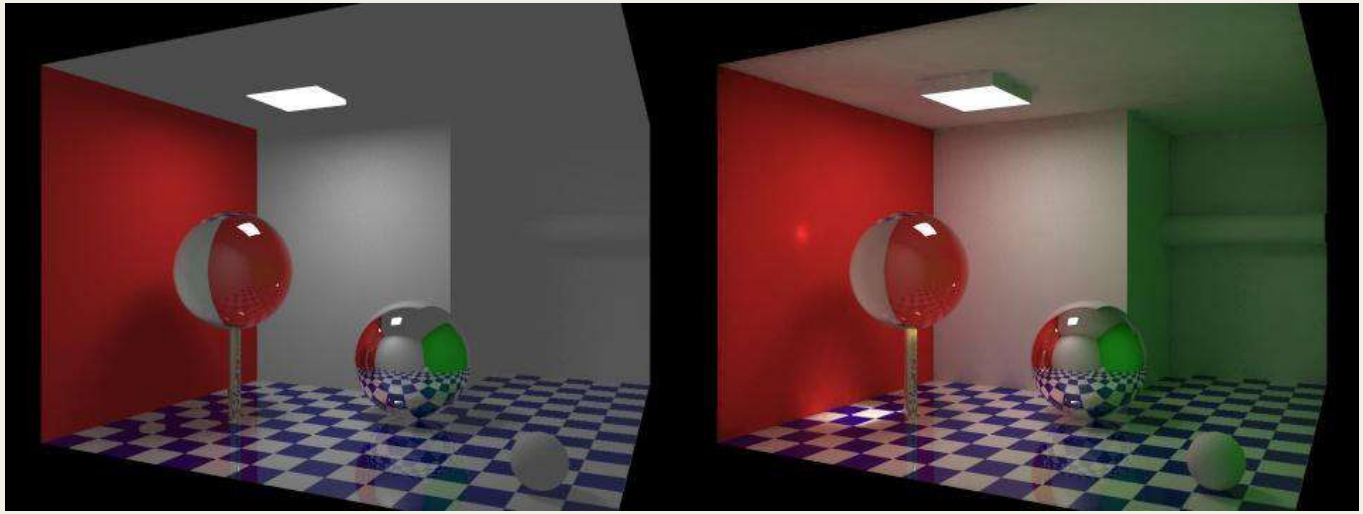


Figure 1: Left: ray casting with shadows (RCS). Middle: Whitted-style ray tracing (WRT). Right: distribution ray tracing (DRT) with 64 samples per pixel. This paper investigates interactive WRT on current hardware and the prospects for interactive DRT on future hardware.

https://graphics.stanford.edu/~boulos/papers/cook_gi07.pdf

Color Bleeding - Review

- In the real world, light doesn't come from just light sources
- Light comes from all visible objects in the world
- Each area chunk of each object acts as a source of light



using only light from the light source

using incoming light from all directions

Path Tracing – Global Illumination

- General Idea: when a ray from the camera intersects with an object, create a new “scatter” ray starting from the intersection point and going out in a randomly sampled direction along the hemisphere
- Recursively bounce the scattered ray until it hits a light - anything it hit on the way should contribute their color for color bleeding!

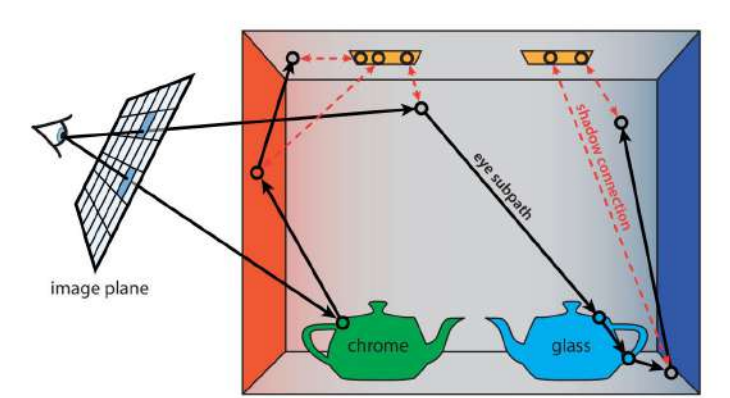
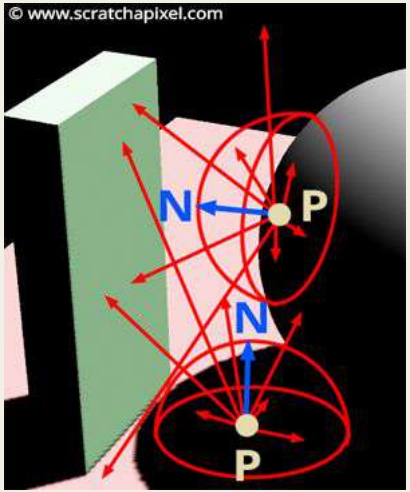


Figure 3.1: An illustration of tracing paths from the eye to the light sources in a Cornell box scene with two teapots.



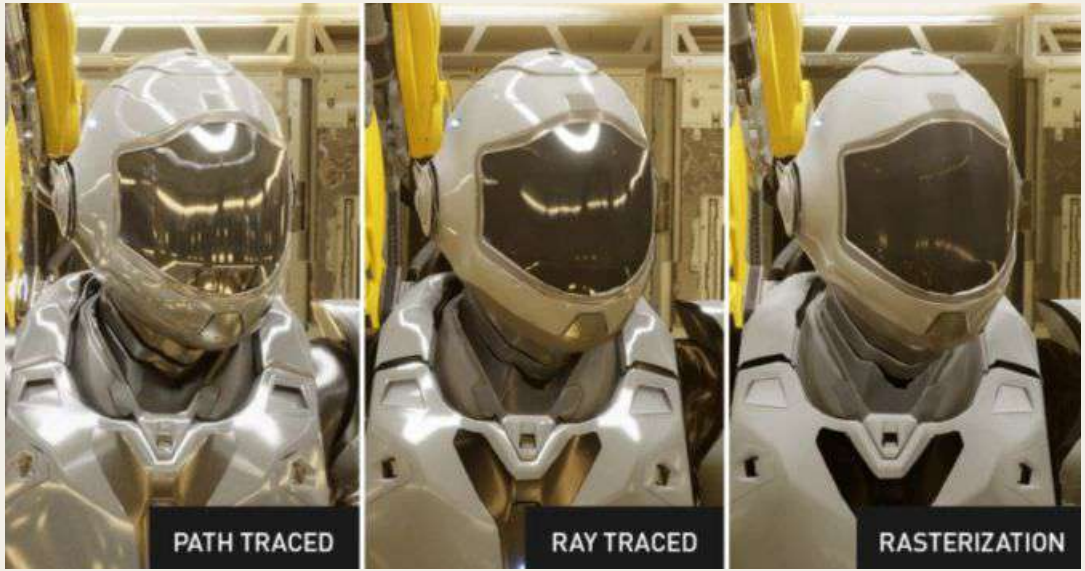
https://graphics.pixar.com/library/PathTracedMovies/paper.pdf

Path Tracing – Global Illumination

- Most realistic result, but can be inefficient – not all scattered rays may lead back to a light within a reasonable number of bounces!

- Often combined with **“smarter” sampling** like importance sampling

- Or approximated with **hybrid methods** like bidirectional ray tracing and photon mapping



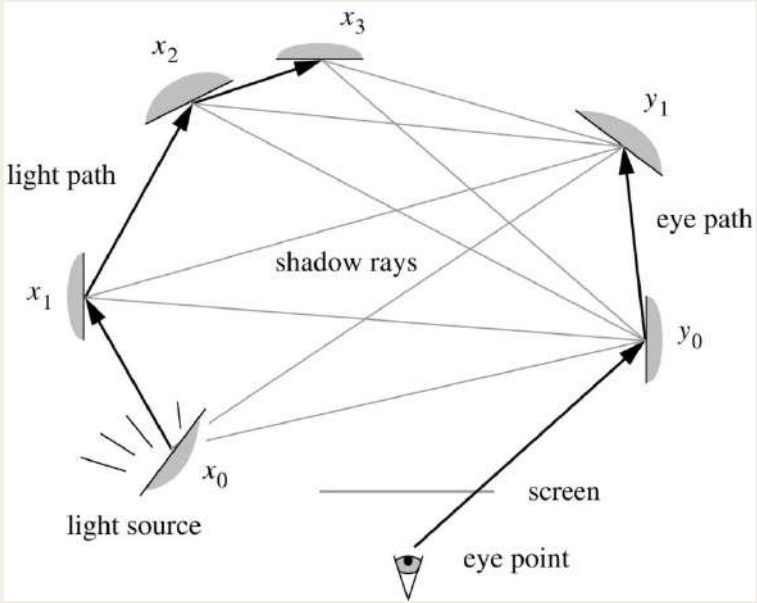
https://blogs.nvidia.com/blog/2022/03/23/what-is-path-tracing/

Hybrid Methods for Global Illumination

- **Bidirectional ray tracing:** send rays from both the camera and the light sources

- **Photon mapping:**
  (1) preprocess step: send “photons” from lights and record a map of where they hit
  (2) gather extra light at points on the map when ray tracing normally

Photon Mapping



Bidirectional Ray Tracing

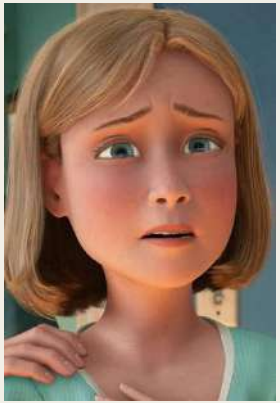
Questions?

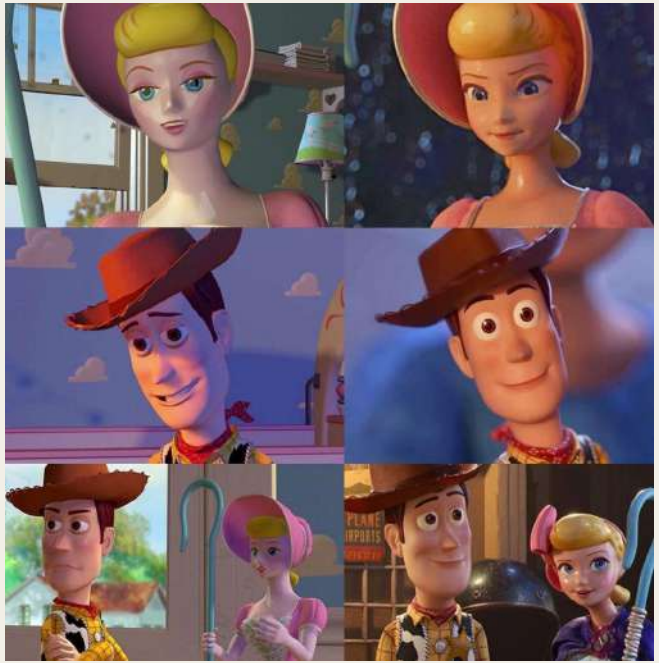
Choosing a technique

- Many animated movies have realistic environment, stylized creatures – are these treated differently/ **due to constraints**?
  - Stylization is often a choice

    in the old days, it was a clever/judicious choice.
    Ex. Toy Story aged well (rasterization, not raytraced. smooth/plastic toys)

    Nowadays, it's often an aesthetical choice
    i.e. we can make it realistic, but it looks nicer if we don't
    Ex. Into the Spiderverse
    extra effort to make CGI look hand-drawn/ non-3D






VARIETY
SHIPPING BAGGAGE
CANDY

Choosing a technique

- Many animated movies have realistic environment, stylized creatures – are these treated differently/ **due to constraints**?
  - Stylization is often a choice

    in the old days, it was a clever/judicious choice.
    Ex. Toy Story aged well (rasterization, not raytraced. smooth/plastic toys)

    Nowadays, it's often an aesthetical choice
    i.e. we can make it realistic, but it looks nicer if we don't
    Ex. Into the Spiderverse
    extra effort to make CGI look hand-drawn/ non-3D

Choosing a technique

- Many animated movies have realistic environment, stylized creatures – are these treated differently/ **due to constraints**?
  - That being said, some things are still hard

    Uncanny valley: why you don't see a lot of attempts at realistic humanoid characters
    Takes a **lot** of effort to make humanoid appearance and motion look "not creepy" or "not game like"
    (won't cover in class, but here are some links for the interested)

https://digitaldomain.com/technology/masquerade-offline-capture/ https://www.unrealengine.com/en-US/metahuman https://www.fxguide.com/fxfeatured/nivellen-may-be-cursed-but-hes-a-winner/

Choosing a technique

- Many animated movies have realistic environment, stylized creatures – are these **treated differently**/ due to constraints?
  - Realism is often based on **capturing** real-world materials/geometry/motion/lighting (computer graphics and computer vision are intertwined)

    Ex. In Snow White and the Huntsman, The Mill used a witness camera to capture lighting to map onto a mirror creature

Choosing a technique

- Many animated movies have realistic environment, stylized creatures – are these **treated differently**/ due to constraints?
  - Photorealism in movies often have an emphasis on **capturing** real-world materials/geometry/lighting (computer graphics and computer vision are intertwined)

    Ex. In Snow White and the Huntsman, The Mill used a witness camera to capture lighting to map onto a mirror creature

See the reflection on the CG creature!
Coming back to this later in lecture …

In the past 7 lectures, we covered a lot of fundamentals

A lot of the fundamentals will continuously appear in different practical considerations (you'll see a lot of references to past lectures here!)

Bihan Liu and Yixin Liu

Yan (Mia) Miao

Enci Liu

Textures

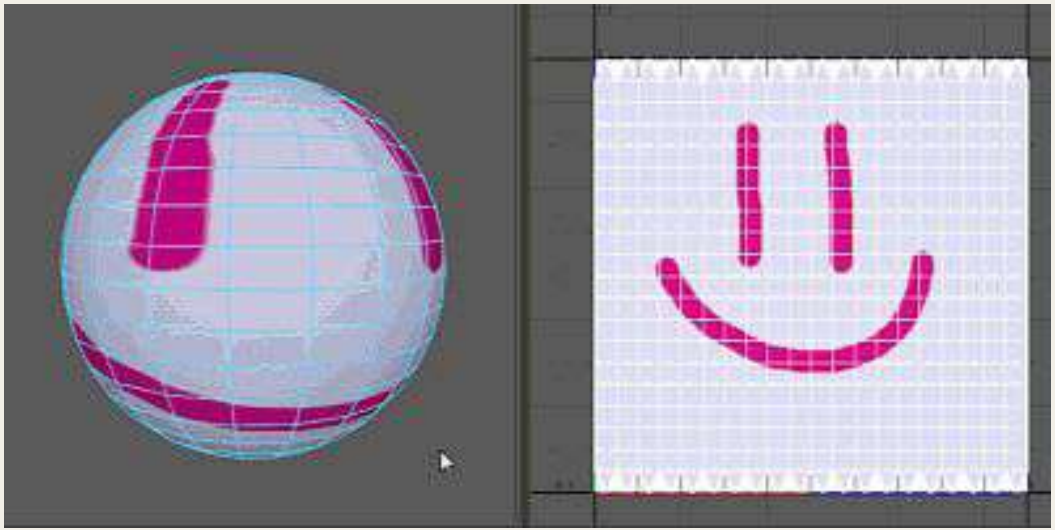
- Idea: we want more complex appearance beyond solid color
- Solution: “Giftwrap” approach

Textures

- Idea: we want more complex appearance beyond solid color
- Solution: “Giftwrap” approach

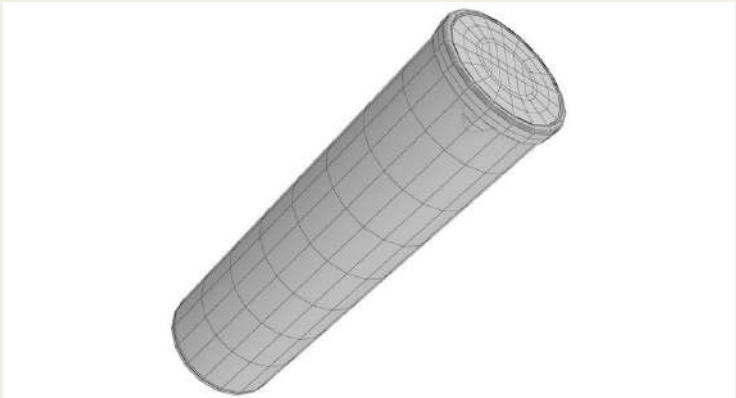
Textures

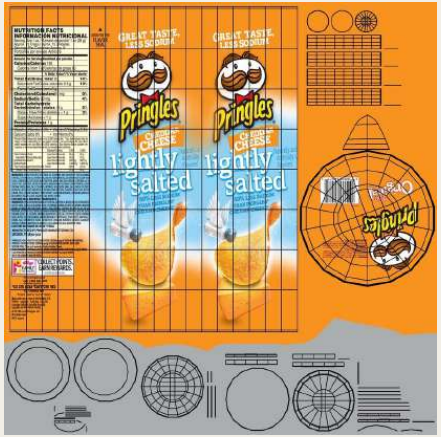
- Idea: we want more complex appearance beyond solid color
- Solution: “Giftwrap” approach:
  parametrize geometric surfaces with a 2D coordinate system,
  so we can paste images (textures) onto geometry



Textures

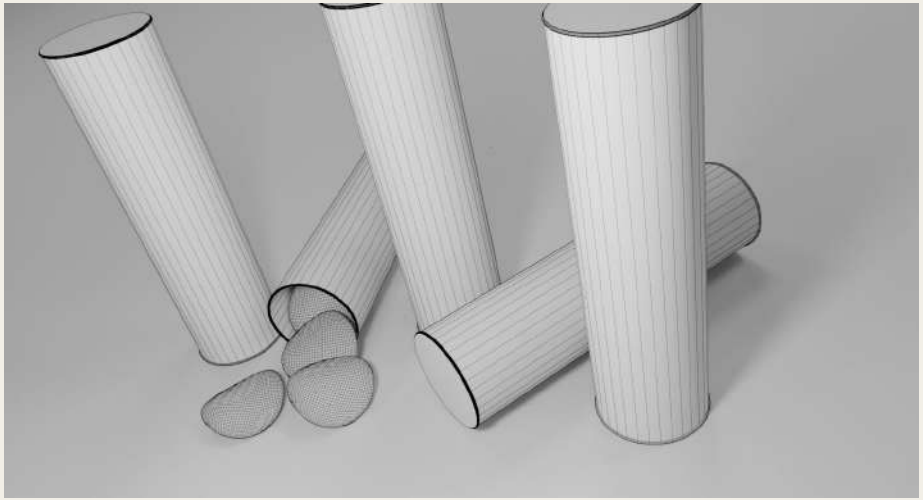
- Idea: we want more complex appearance beyond solid color
- Solution: “Giftwrap” approach:
  parametrize geometric surfaces with a 2D coordinate system,
  so we can paste images (textures) onto geometry





Different way of thinking about this…

- Rasterization framework (splatting) -> Raytracing framework (point-based)
- Moving away from “solid color” materials by creating many BxDFs (BRDF/BTDF/BSSRDF…) to achieve each separate look is inefficient!
- Idea: Single **spatially varying** BxDF – Recall lecture 5 with BRDF$\left(\lambda, \omega_i, \omega_o, u, v\right)$

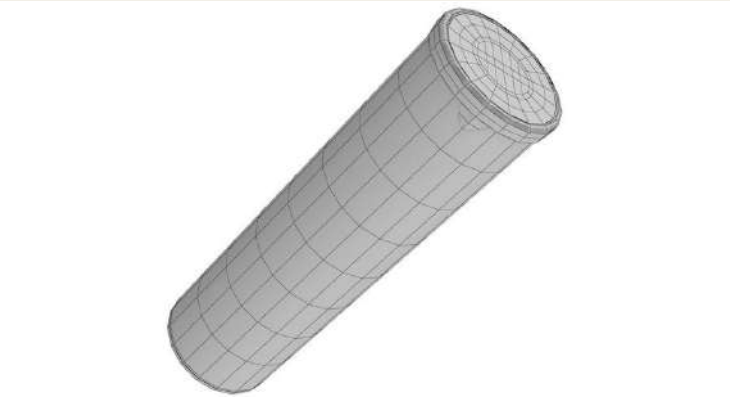


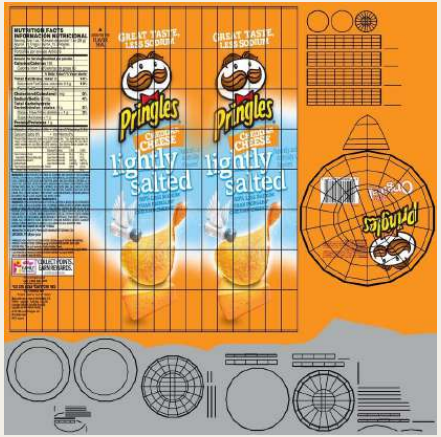
https://3docean.net/item/pringles-can-shout-potato-chips-3d-model/27058795

Textures

- More general “rendering equation”-based way to frame this:
  define a 2D function on the surface,
  where the function outputs coordinates that we can use to lookup BxDF colors/parameters from a 2D grid

$$L_o(\omega_o) = \int_{i \in in} BRDF(\omega_i, \omega_o) dE_i(\omega_i)$$





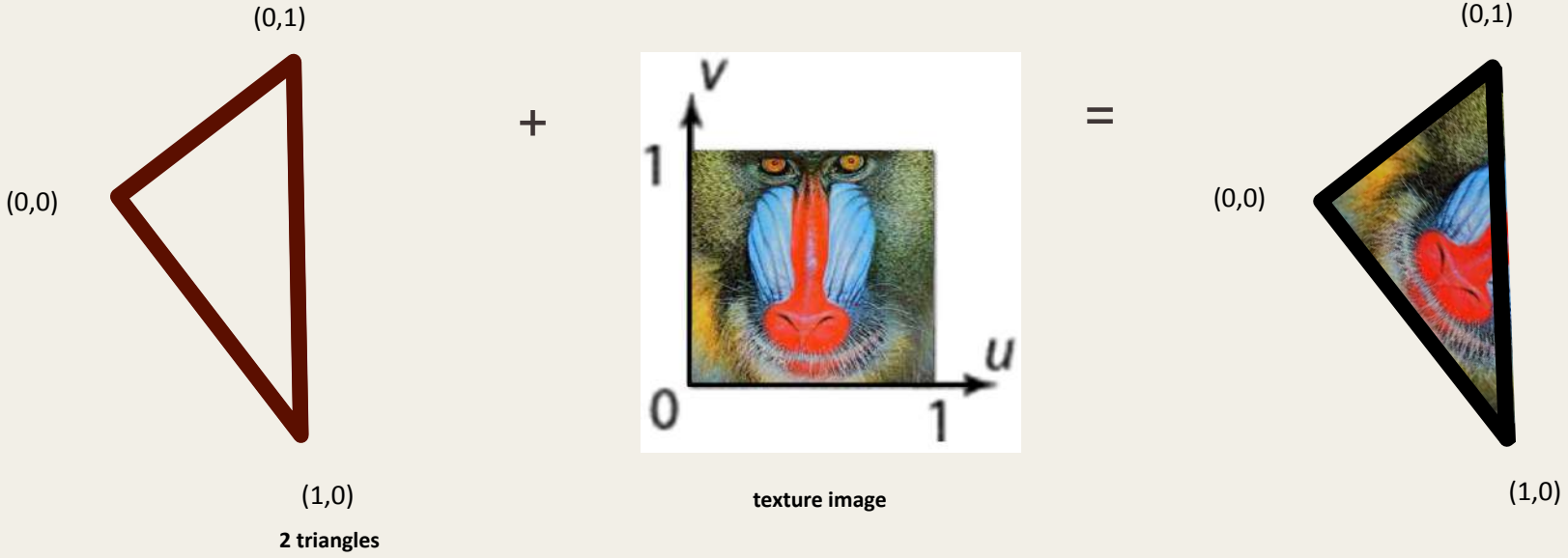
Lecture Outline

- Texture Mapping: UV Coordinates and Creation
- Texture Lookup: Interpolation and Sampling
- Texture Frameworks: One for each BxDF parameter
- Normal Mapping: “Hacking” geometry with textures

- Texture Acquisition: Art, Capture, Procedural
- Environment Mapping and Sky Boxes

Formal Approach

- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties
- (2D grid lookup table for RGB colors = an image/texture)!



OBJ file

- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties
- (2D grid lookup table for RGB colors = an image/texture)!

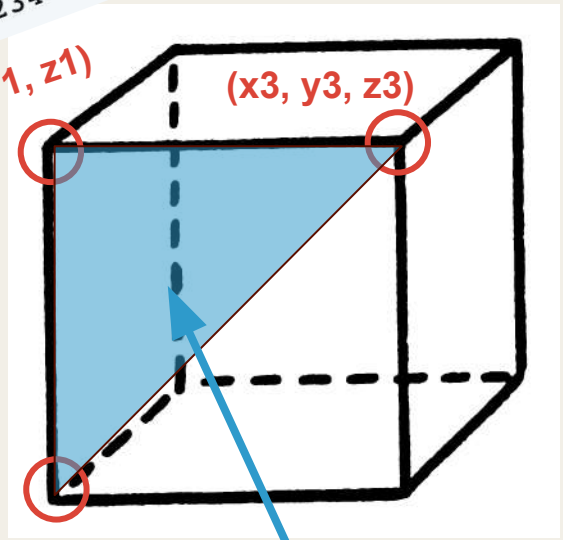
```
v 0.123 0.234 0.345
```

```
vt 0.500 1
```

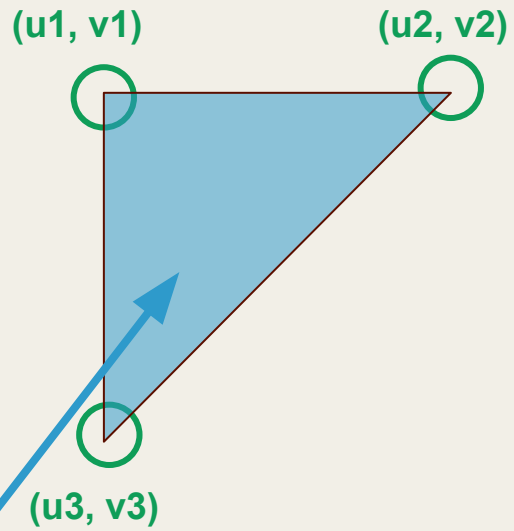
```
f v1 v2 v3 ....
```

```
f v1/vt1 v2/vt2 v3/vt3 ...
```

```
v 0.123 0.234 0.345
```



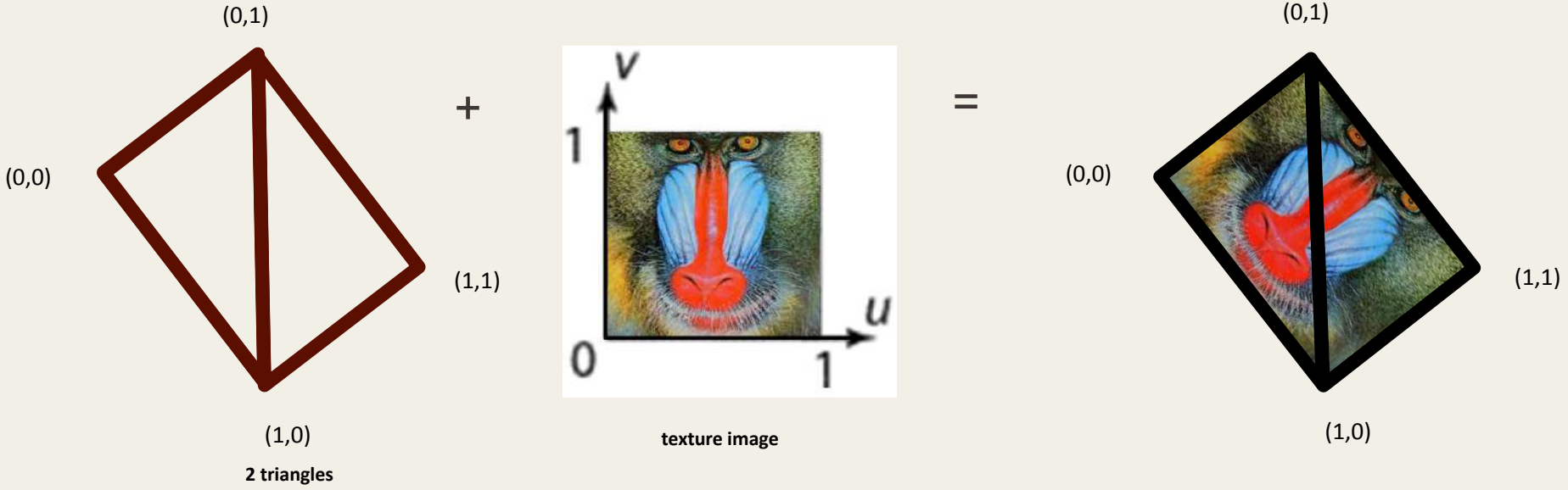

```
vt 0.500 1
```




```
f v1/vt1 v2/vt2 v3/vt3 ...
```

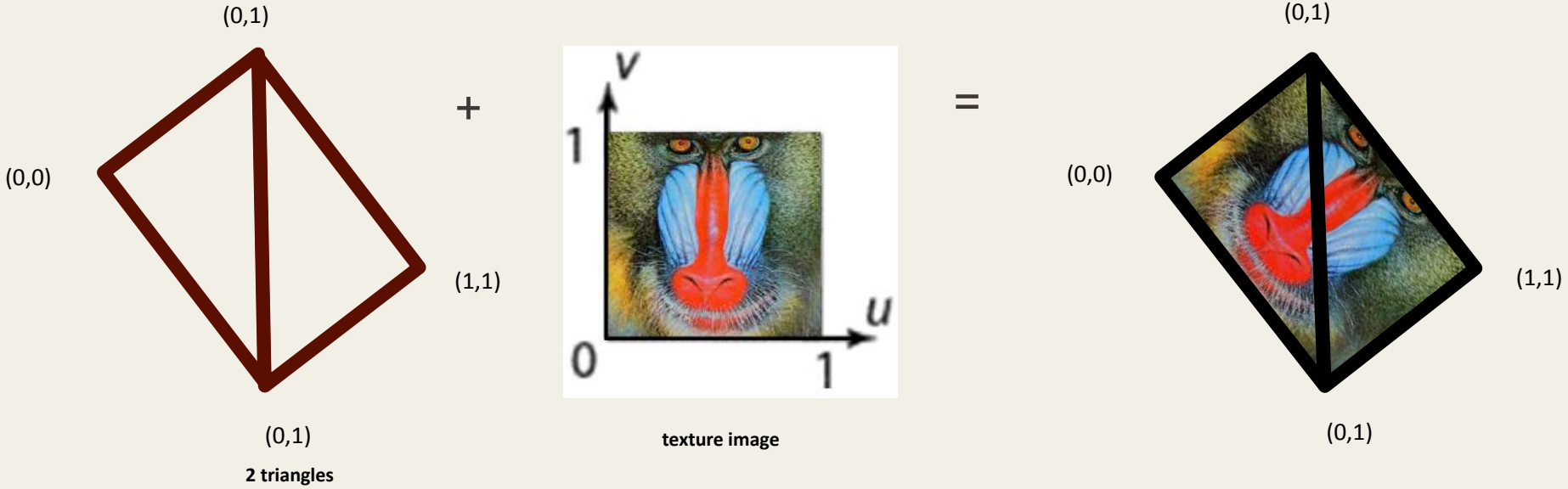
Formal Approach

- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties
- (2D grid lookup table for RGB colors = an image/texture)!



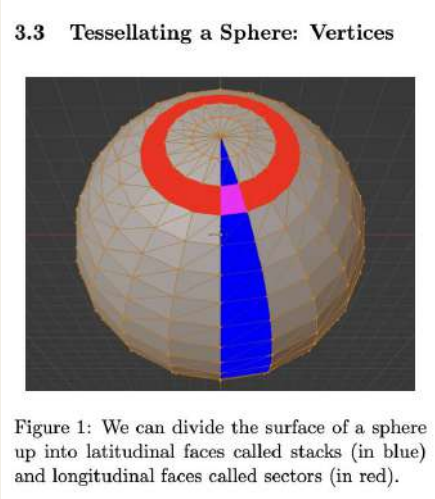
Some details

- How do we get 2D coordinates?
- How do we use the lookup table?
- How do we get the lookup table?



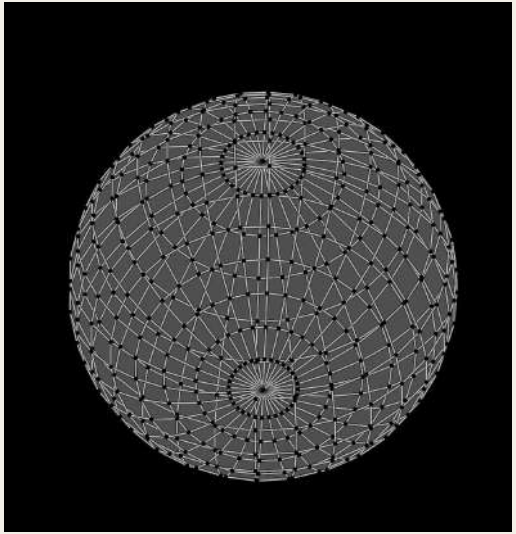
Assigning UV coordinates

- Can construct analytical functions for **implicit geometry** (sphere, cylinder, plane… ex. Homework 1)
- Many options even for a single type of geometry



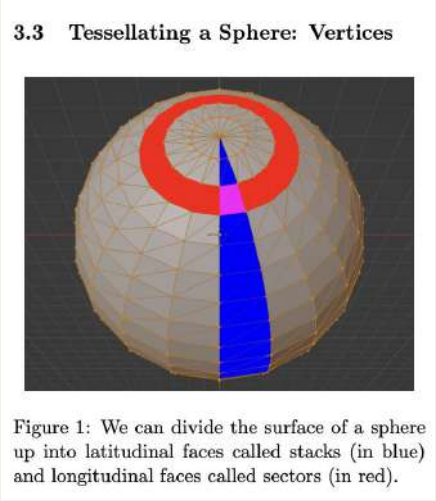
3.3 Tessellating a Sphere: Vertices

Figure 1: We can divide the surface of a sphere up into latitudinal faces called stacks (in blue) and longitudinal faces called sectors (in red).



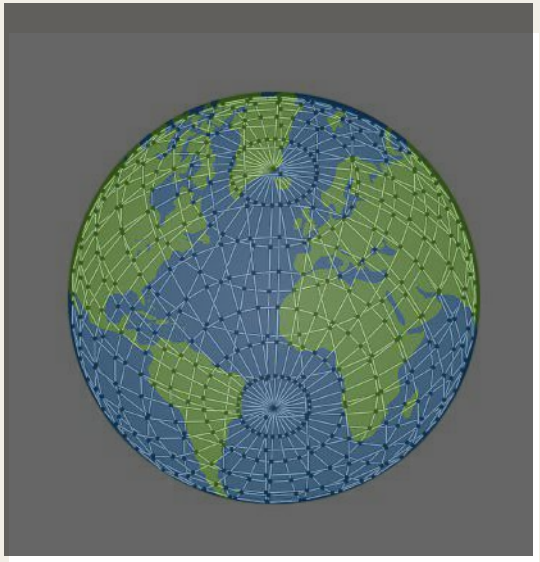
Assigning UV coordinates

- Can construct analytical functions for **implicit geometry** (sphere, cylinder, plane… ex. Homework 1)
- Many options even for a single type of geometry



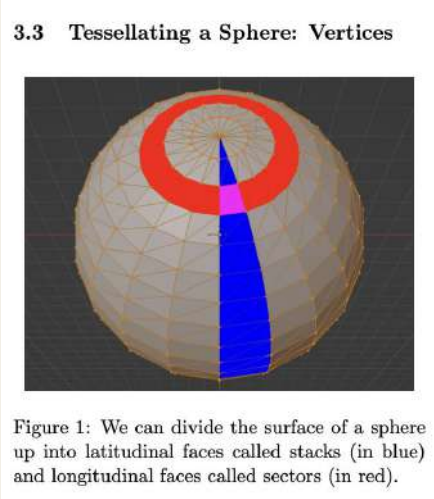
3.3 Tessellating a Sphere: Vertices

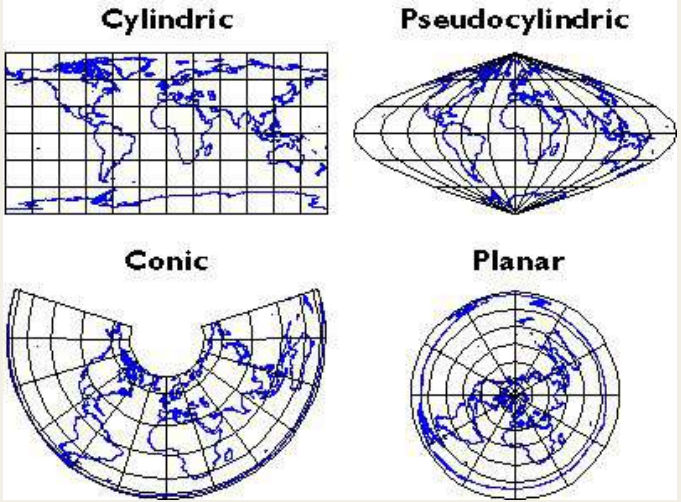
Figure 1: We can divide the surface of a sphere up into latitudinal faces called stacks (in blue) and longitudinal faces called sectors (in red).

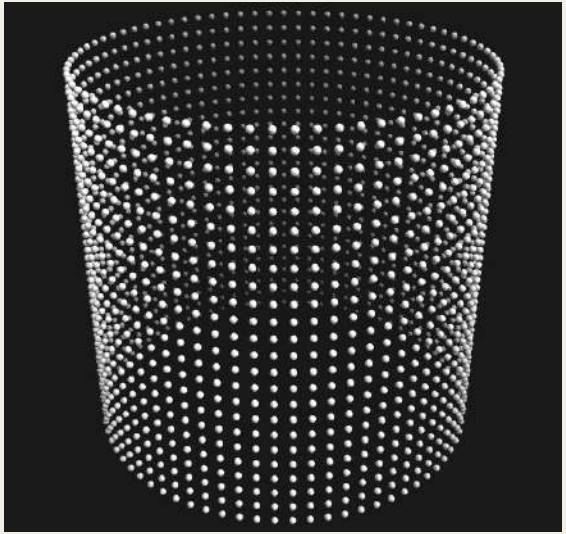


Assigning UV coordinates

- Can construct analytical functions for **implicit geometry** (sphere, cylinder, plane… ex. Homework 1)
- Many options even for a single type of geometry

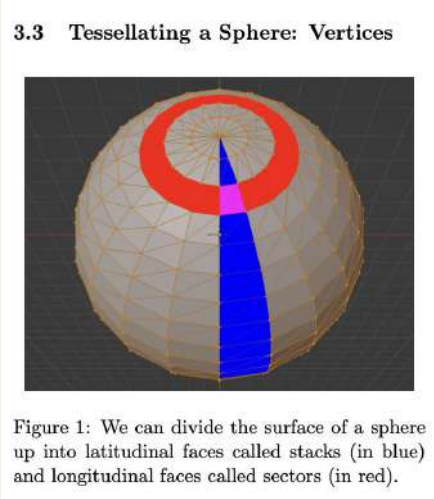


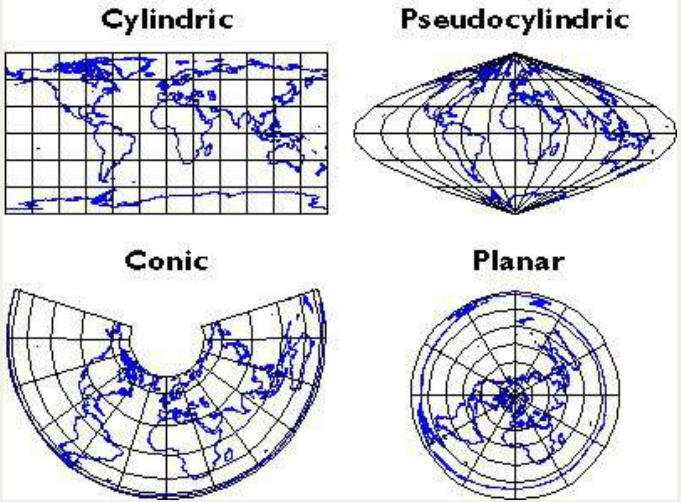


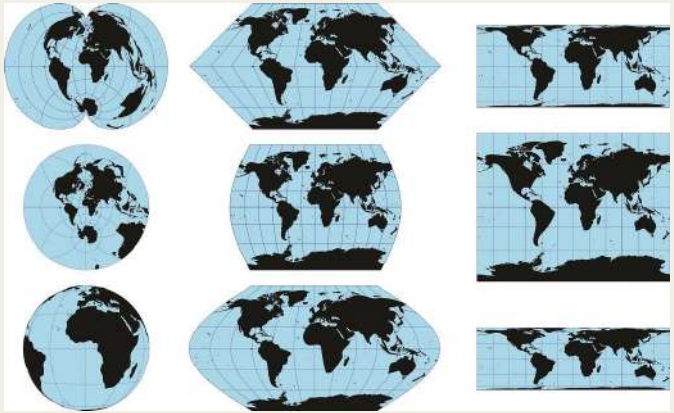


Assigning UV coordinates

- Can construct analytical functions for **implicit geometry** (sphere, cylinder, plane… ex. Homework 1)
- Many options even for a single type of geometry

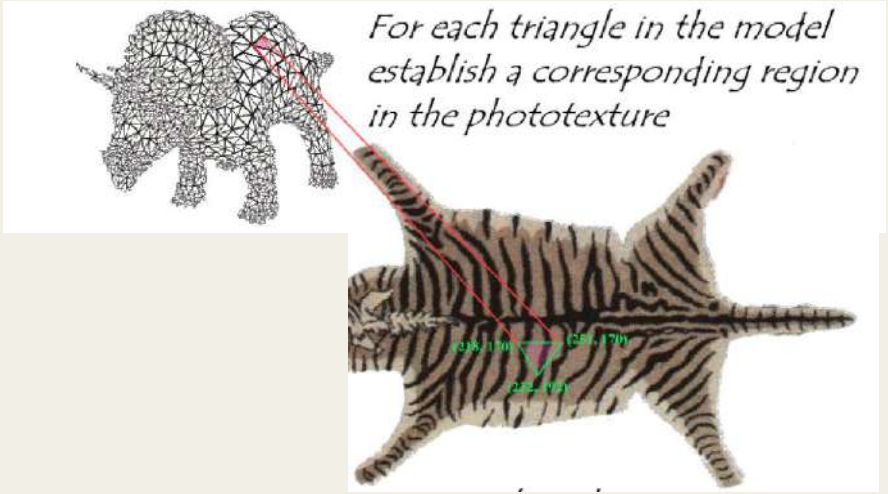






Assigning UV coordinates

- Define per-polygon for **explicit geometry** (polygon meshes)
- Recall simple definition of triangle mesh (Lecture 2):
  List of vertex positions
  + List of indices into vertex list
  $[(vi_1, vi_2, vi_3)]$

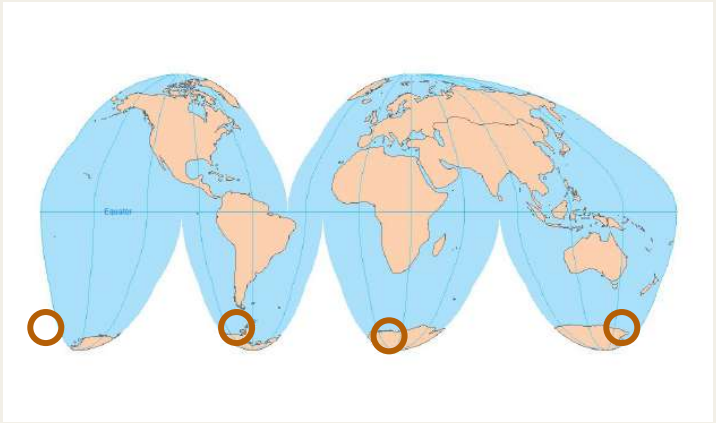


- Now, we have
  List of vertex positions
  + List of UV coordinates
  + List of indices into vertex list +uv list $[([vi_1,ti_1], [vi_2, ti_2], [vi_3,ti_3])]$
  Note: a single vertex might map to multiple texture coordinates

Assigning UV coordinates

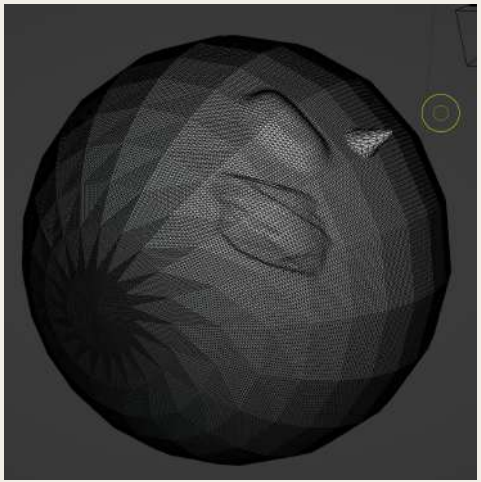
- List of vertex positions
  + List of UV coordinates
  + List of indices into vertex list +uv list ([vi_1,ti_1], [vi_2, vi_2], [vi_3,ti_3])
  Note: a single vertex might map to multiple texture coordinates
  Can't simply assign a UV coordinate to each vertex position.

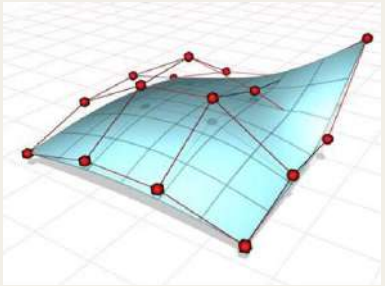
  Simple example:
  UV mapping a globe
  (look at the south pole!)

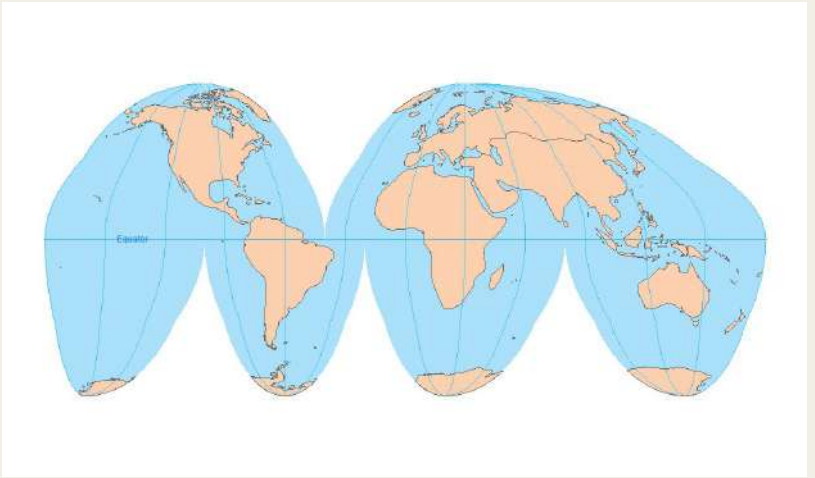


UV Unwrapping

- Recall Lecture 2 : constructing geometric objects.
  Manually picking per-vertex for explicit objects is **intractable**
- Create seams, then project/flatten/warp (“unwrap” the object)
- **Blender Demo #1! [Homework Content]**

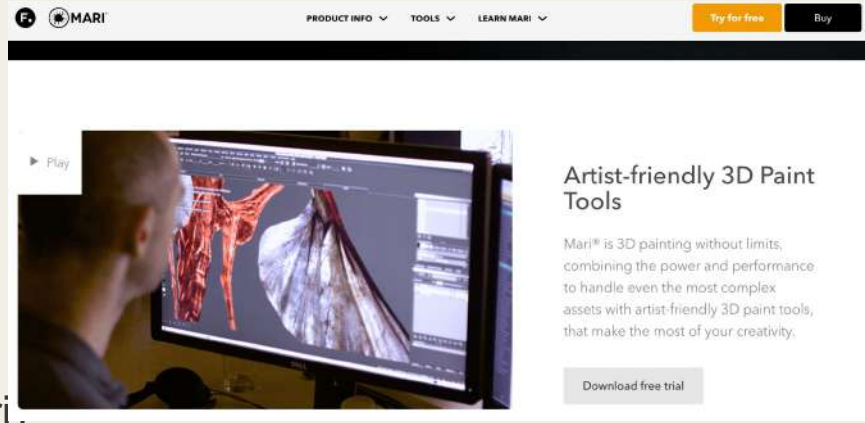






UV Unwrapping

- Requires texturing tools:
  - computational geometry algorithms
  - specialized user interfaces
  - artist expertise

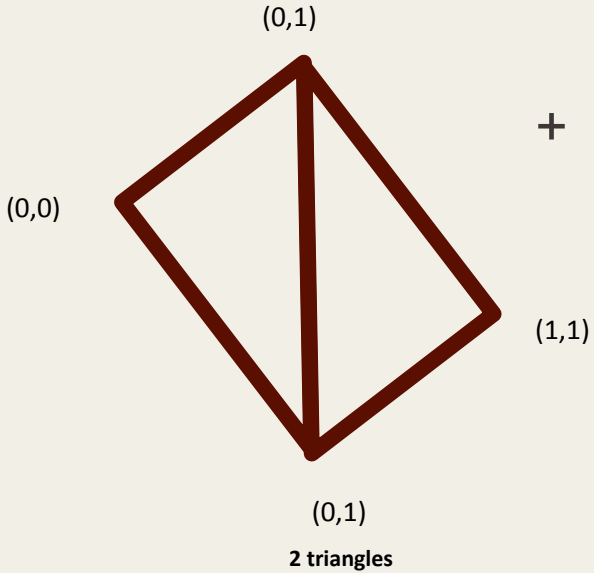


- Specialized Software (Lecture 1)
  - All-in-1 (Blender) vs Specialized (Mari)
  - You probably have a sense by now…
    how hard to it is build all-in-1 graphics tools
  - Just “rendering” itself requires a lot of infrastructure (Lectures 5-7)

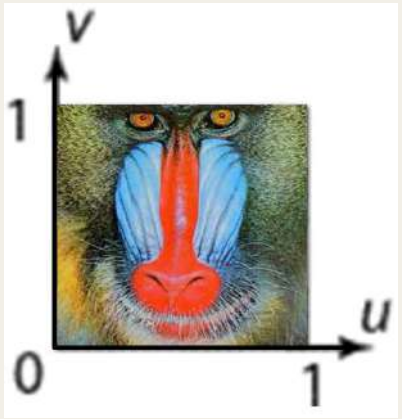
- But again, Blender/Maya is more than enough most of the time

Some details

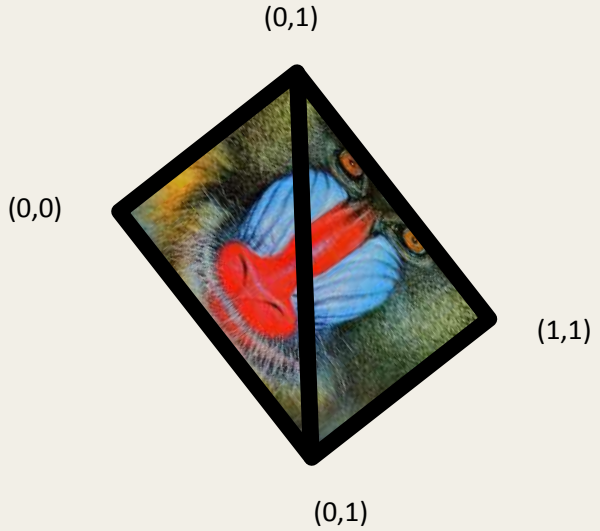
- How do we get 2D coordinates? **(per-vertex coordinates -> surface?)**
- How do we use the lookup table?
- How do we get the lookup table?



+



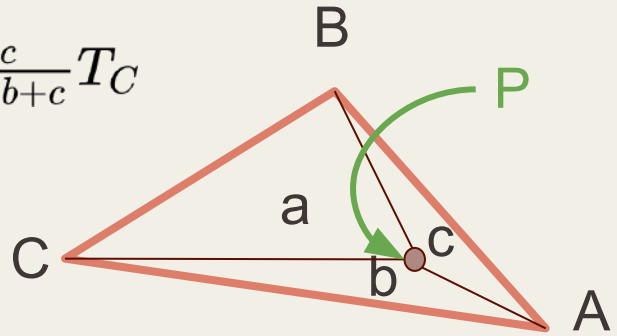
=

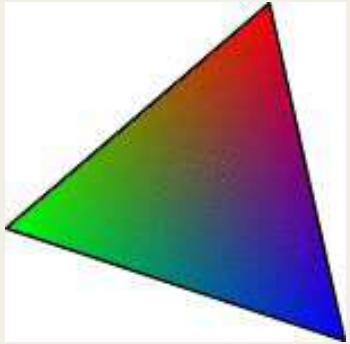


Barycentric Interpolation

- Method to spatially interpolate between vertices to triangle interior
- Recall Lecture 3 (vertex -> fragment shaders)
- Note:
  - for raytracing, correct barycentric values in **3D space**
  - for rasterization, naive algorithm on **perspective projection** (Lecture 4) actually gives incorrect values

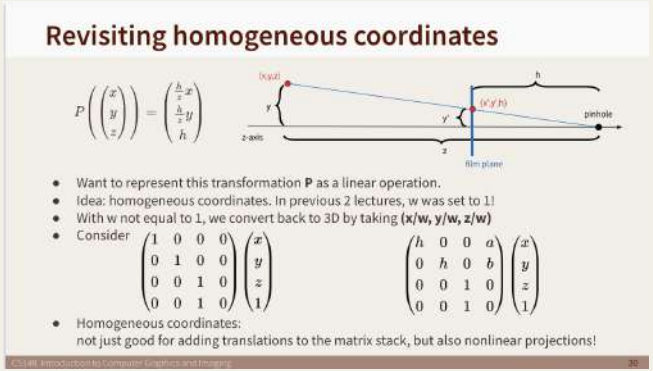
$$T_p = \frac{a}{a+b+c}T_A + \frac{b}{a+b+c}T_B + \frac{c}{a+b+c}T_C$$

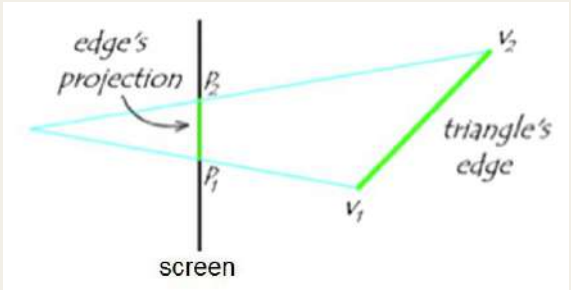


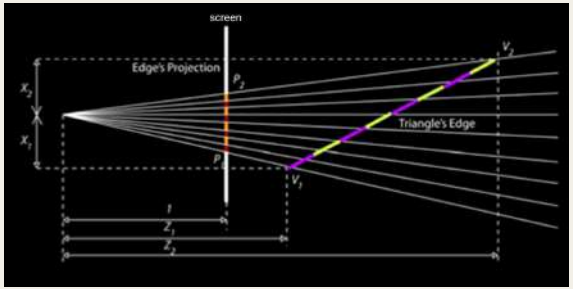


Barycentric Interpolation

- for raytracing, correct barycentric values in **3D space**
- for rasterization, naive algorithm with **perspective projection** gives incorrect values
  (perspective projection divides by z - nonlinearly changes the triangle's shape!)





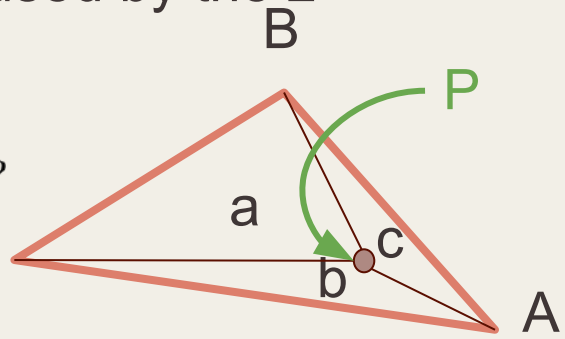


Screen Space Barycentric Interpolation

- Perspective correct interpolation: high-level/hand-wavy explanation
  is that we need to deal with nonlinearity caused by the z component (depth) division

$$T_p = \left(\frac{a}{a+b+c} \frac{T_A}{z_A} + \frac{b}{a+b+c} \frac{T_B}{z_B} + \frac{c}{a+b+c} \frac{T_C}{z_C} \right) z_P$$

$$z_p = \frac{a}{a+b+c} z_A + \frac{b}{a+b+c} z_B + \frac{c}{a+b+c} z_C$$



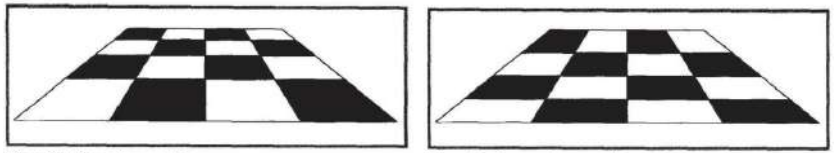


Figure 2a. Correct perspective.

Figure 2b. Incorrect perspective.

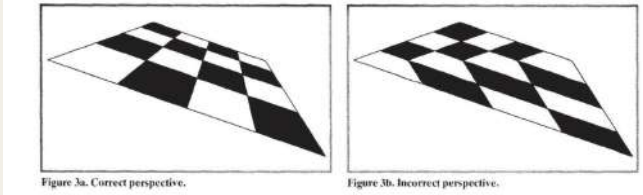


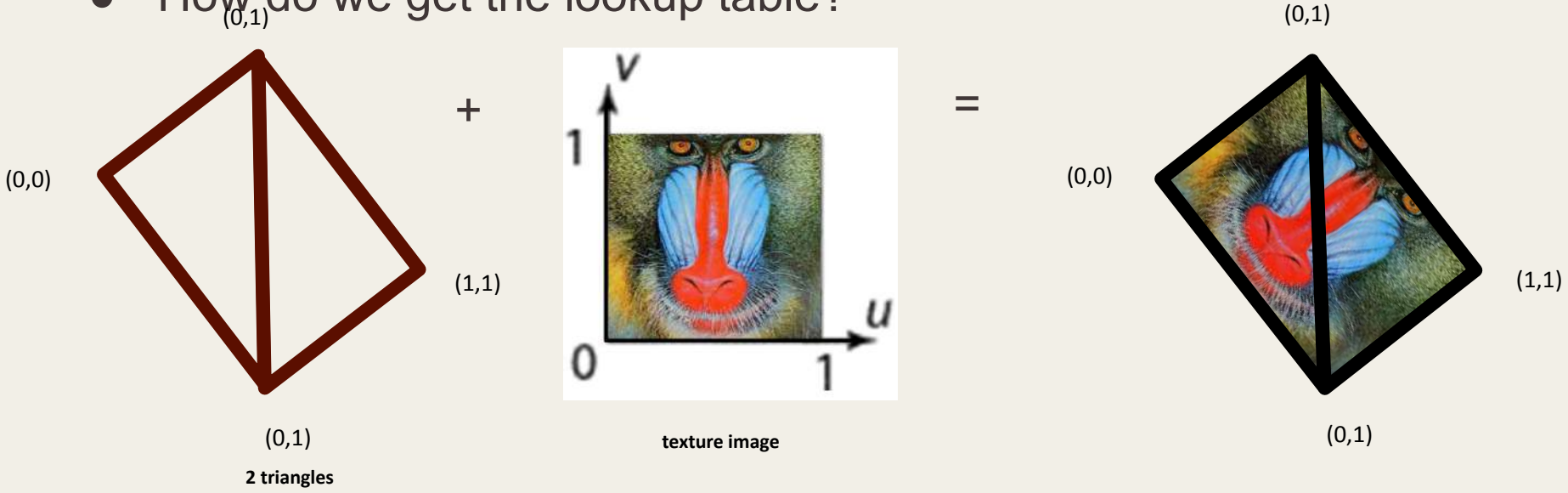
Figure 3a. Correct perspective.

Figure 3b. Incorrect perspective.

See https://core.ac.uk/download/pdf/265963151.pdf (written in 1992, by James Blinn of the Blinn-Phong reflection model) for a great explanation + more homogenous coordinates (Lecture 2 and 4)!

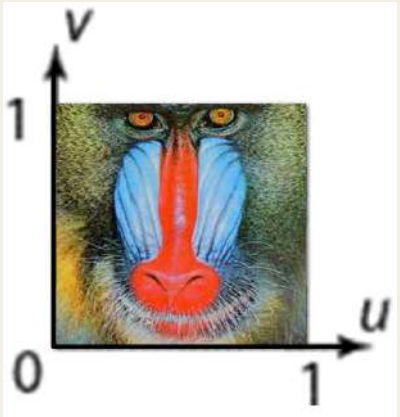
Some details

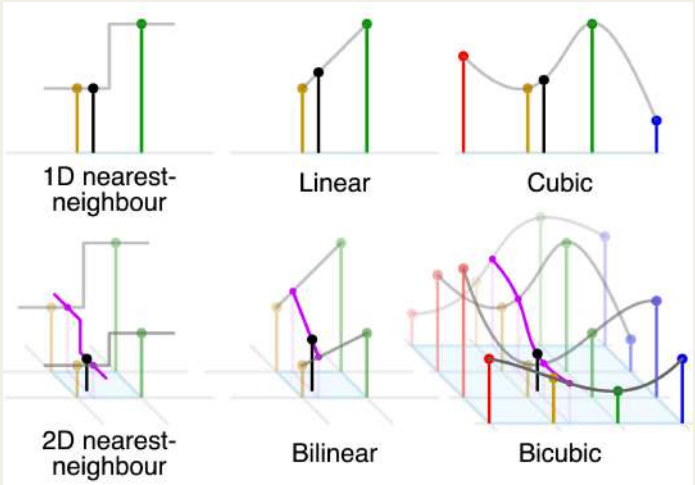
- How do we get 2D coordinates? (per-vertex coordinates -> barycentric interpolation)
- **How do we use the lookup table?**
- How do we get the lookup table?

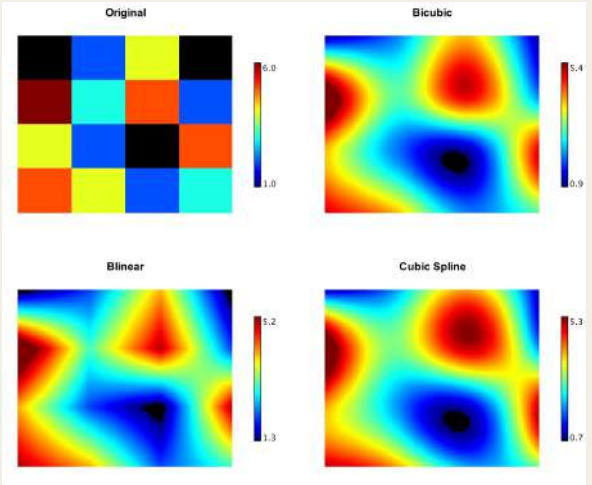


UV Coordinates to 2D grid Lookup

- How to go from UV coordinates (floating point) to 2D pixel grid lookup?
  2D grid is quantized: need interpolation.
  Ex. with a 100 x 100 pixel RGB texture,
  UV coordinate of (0.253,0.75) maps to (**25.3**,75) in image space.
  Many ways to resolve the 25.3 – splines to the rescue! (Lecture 2)
  Note: will talk about UV coordinates >1 later.





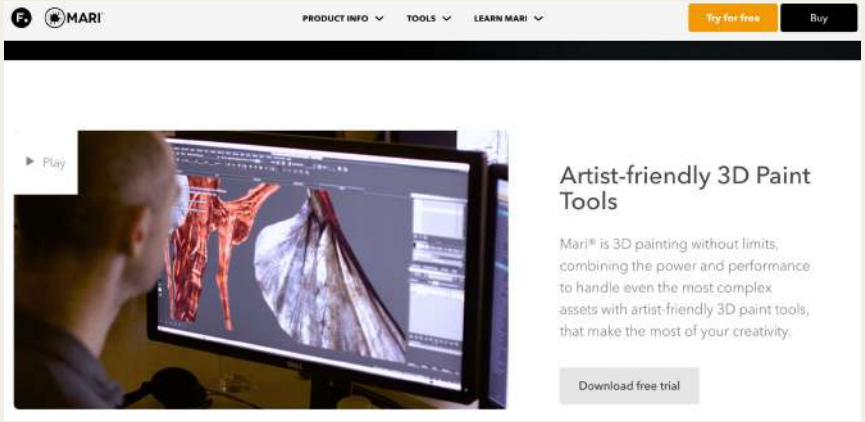


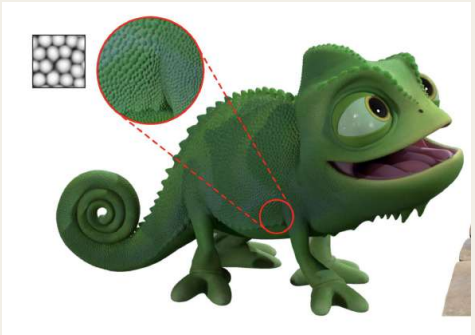
UV Coordinates to 2D grid Lookup

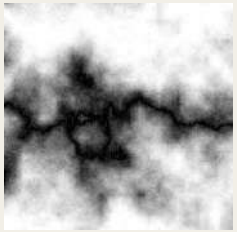
- Additionally: high resolution images may have aliasing artefacts
  Ex. with a 10000 x 10000 pixel RGB texture,
  UV coordinates of geometry rendered in two neighboring pixels might map to pixels very far away
- Idea 1: super sampling the render then filtering/smoothing (Lecture 7)
- Idea 2: Store textures at multiple scales
  resolution-dependent lookup (ex. MIP maps)
  Optimal is 1 pixel to one texture pixel (texel)

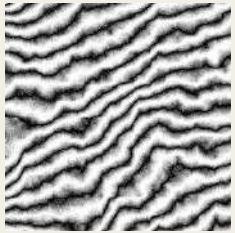
Texture Acquisition: Art, Capture, Procedural

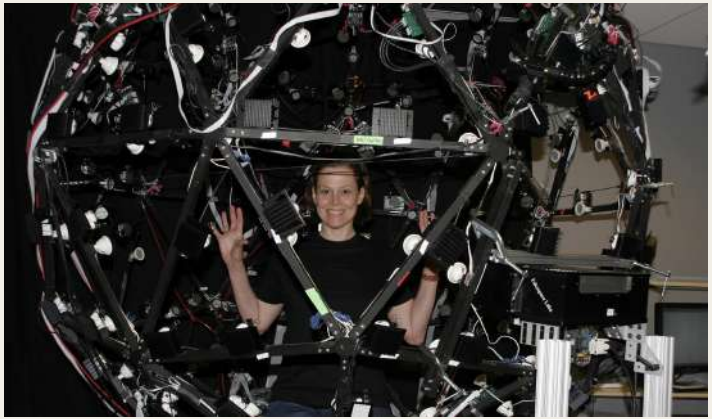
- Artist crafted textures
  (Aided with specialized software)
- Real-world captured data
  (Specialized hardware,
  need artist cleanup as well)
- **Procedural textures**
  (Crafted for “generic use”)





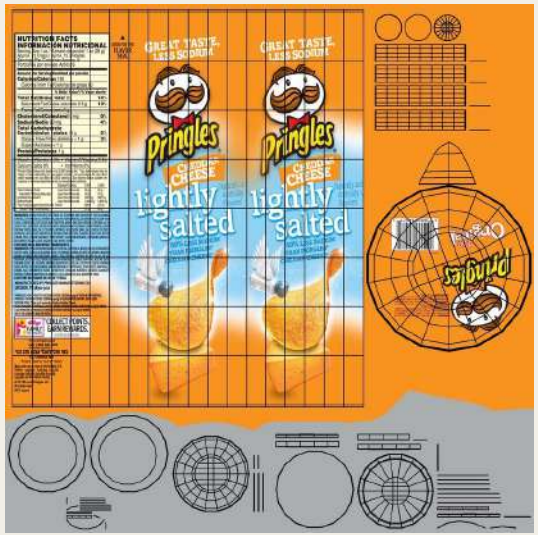


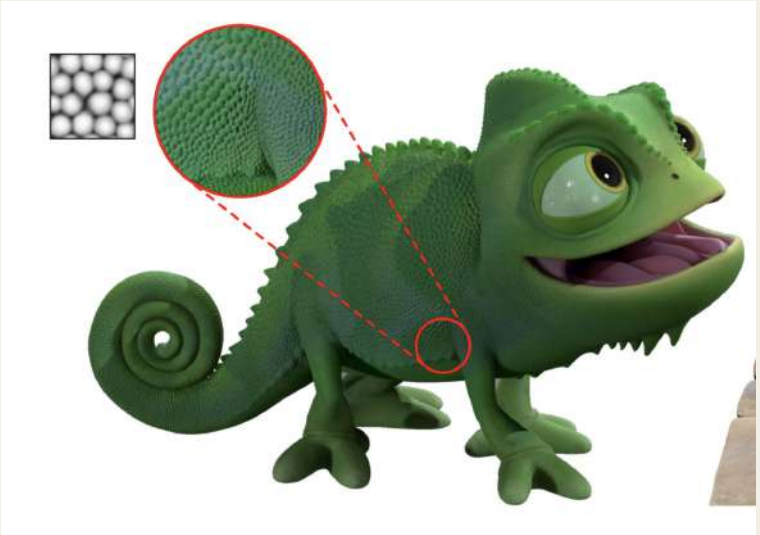




Generalized seamless textures

- Besides textures crafted for specific models,
  There are textures that are designed to be non-model specific

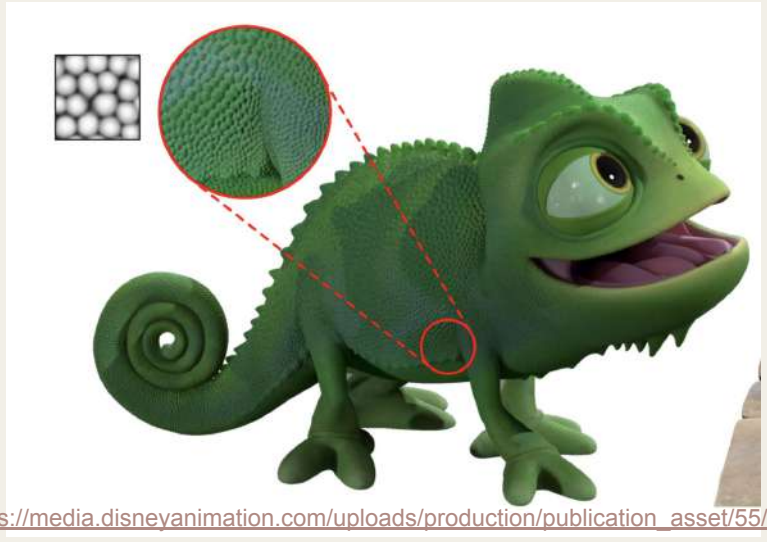




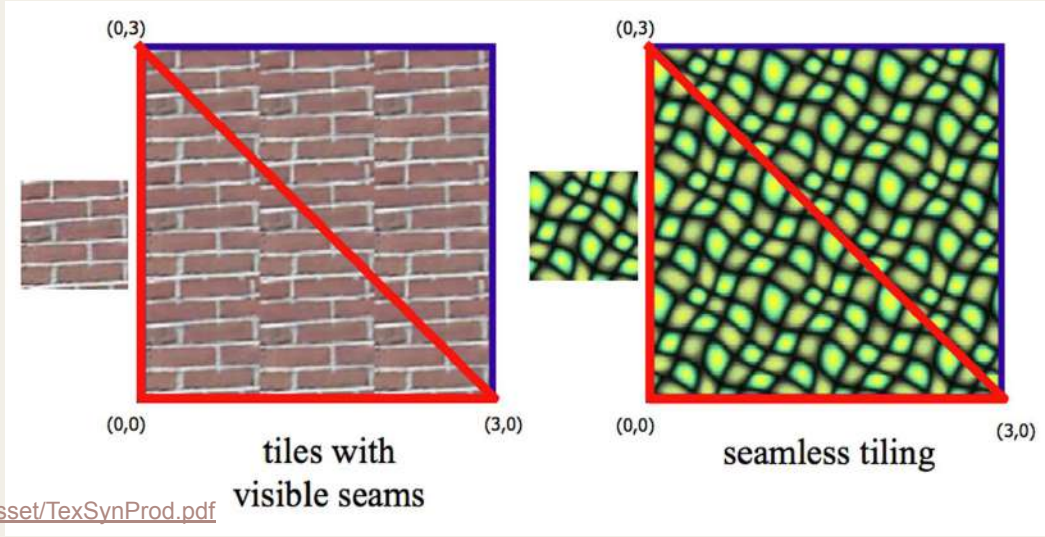
https://media.disneyanimation.com/uploads/production/publication_asset/55/asset/TexSynProd.pdf

Generalized seamless textures

- Can add interesting patterns or repeating details via **seamless** patterns
  - **UV coordinates don't have to be 0-1** : can "wrap around" for high resolution detail without needing high resolution textures
  - Ex. UV coordinate (1.2,2.5) maps to (0.2,0.5) on the lookup grid
- **Blender Demo #3 [Homework Content]**

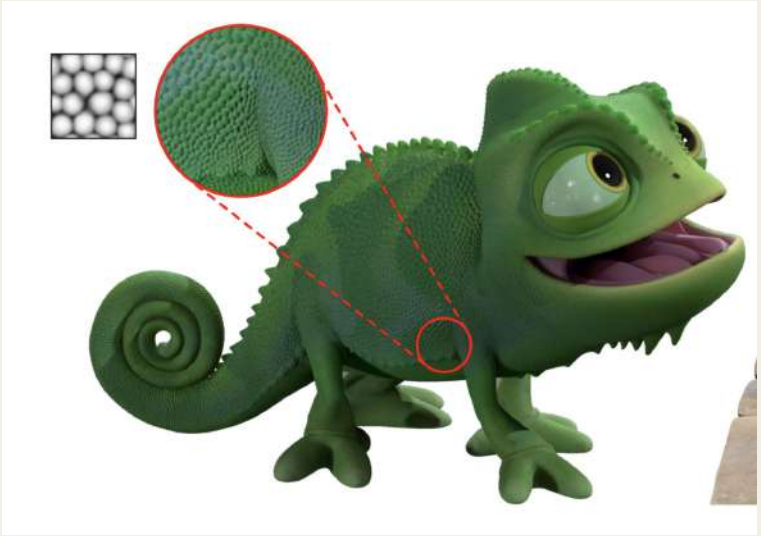


https://media.disneyanimation.com/uploads/production/publication_asset/55/asset/TexSynProd.pdf



Generalized seamless textures

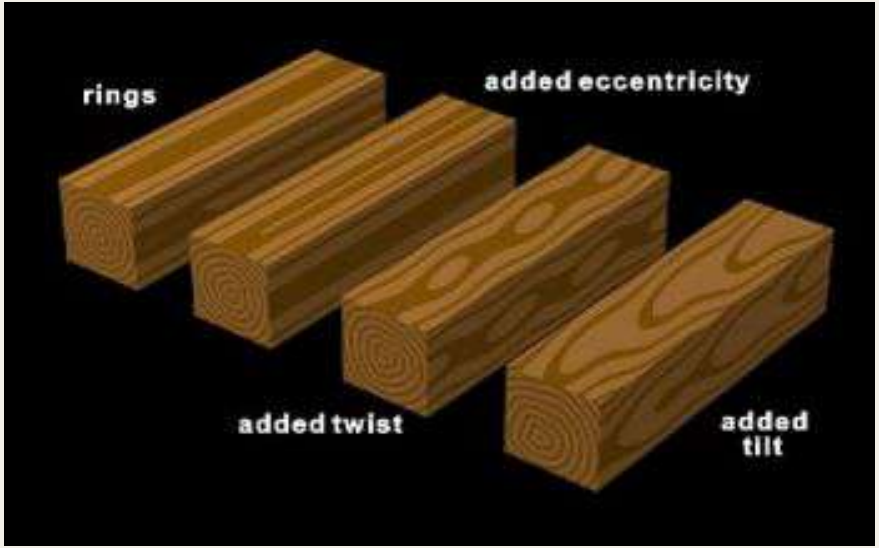
- Can add tiny repeating details via **seamless** patterns
  Can be artist generated OR procedurally generated



Can try these out on the homework! https://renderman.pixar.com/pixar-one-thirty

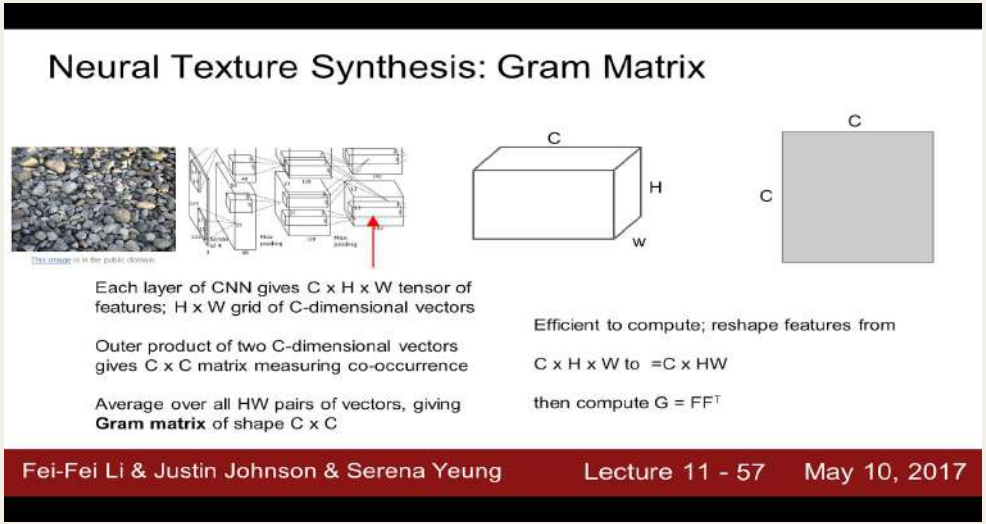
Procedural textures

- Can be explicitly modeled (ex. 3D wood textures)
  Note: we use 2D textures a lot because we deal with surfaces most of the time.
  But 3D texture lookups are perfectly valid as well
- Can be modeled with random noise (Perlin/Simplex Noise)
- Can be created with ML/AI models



Procedural textures

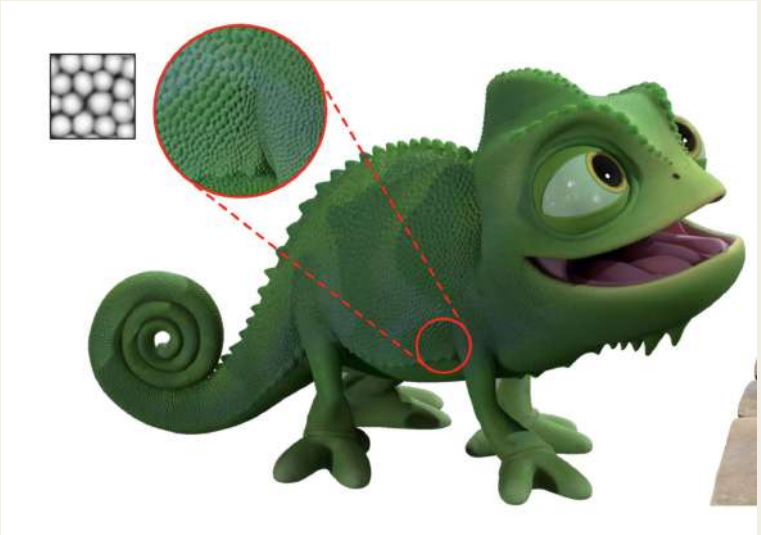
- Can be explicitly modeled (ex. 3D wood textures)
  Note: we use 2D textures a lot because we deal with surfaces most of the time.
  But 3D texture lookups are perfectly valid as well
- Can be modeled with random noise (Perlin/Simplex Noise)
- Can be created with ML/AI models



https://arxiv.org/pdf/1505.07376.pdf

UV Unwrapping

Blender Demo #2! [Homework Content]



Lecture Outline

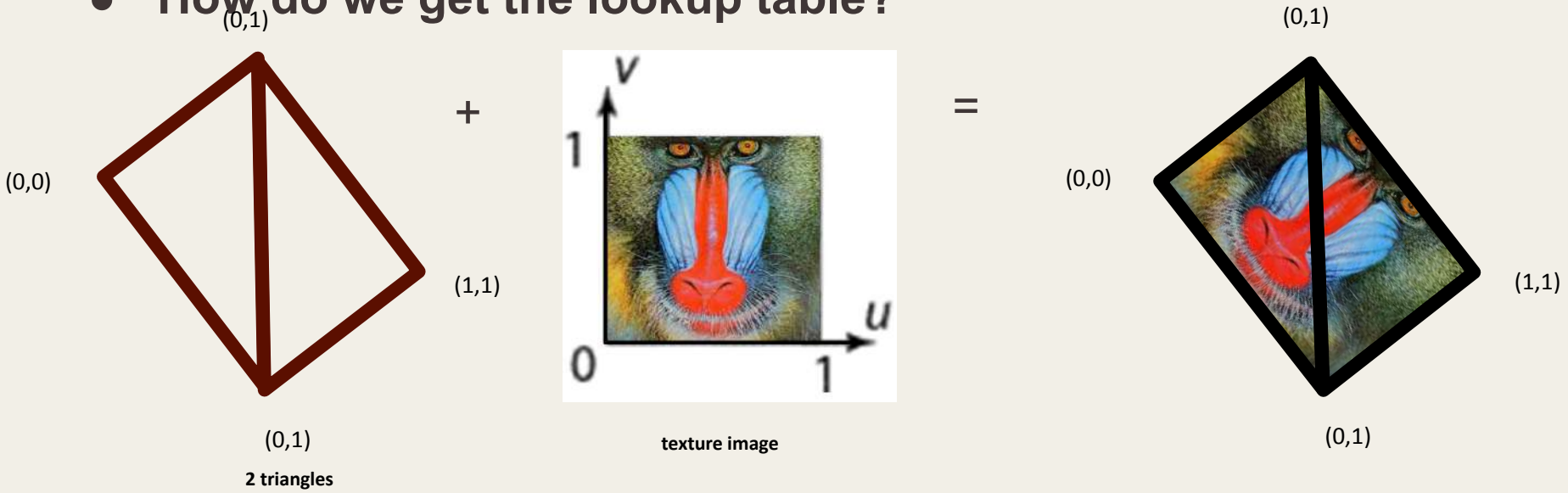
- Texture Mapping: UV Coordinates and Creation
- Texture Lookup: Interpolation and Sampling
- Texture Frameworks: One for each BxDF parameter
- Normal Mapping: “Hacking” geometry with textures

- Texture Acquisition: Art, Capture, Procedural
- Environment Mapping and Sky Boxes

Questions?

Some details

- How do we get 2D coordinates? (per-vertex coordinates -> barycentric interpolation)
- How do we use the lookup table? (need to interpolate/sample)
- **How do we get the lookup table?**



Some details

- How do we get 2D coordinates? (per-vertex coordinates -> barycentric interpolation)
- How do we use the lookup table? (need to interpolate/sample)
- **How do we get the lookup table?**

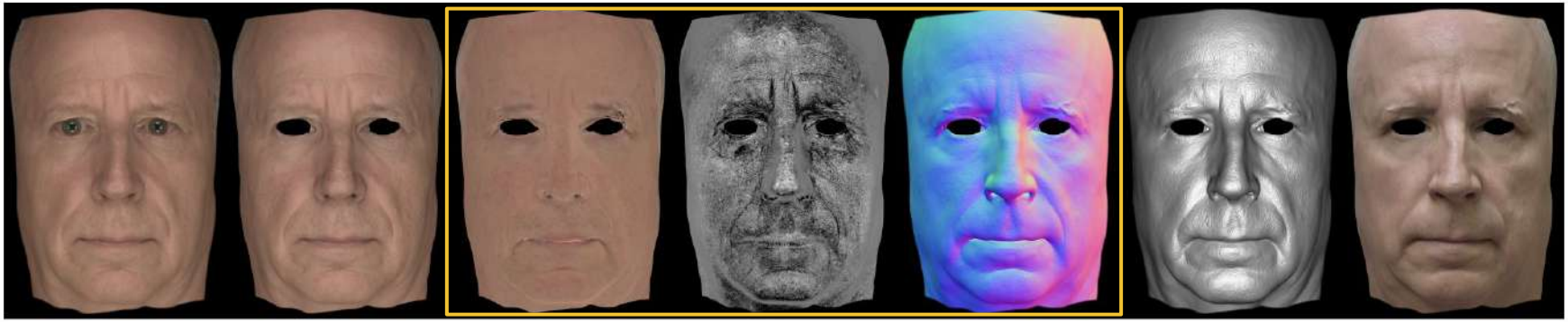


Fig. 1. We present a method to capture complete facial geometry and appearance from a *single* exposure. From left to right: one input image, our matching render, diffuse albedo, specular intensity, normals, high resolution geometry, and a realistic re-render under a different environment map.

Obtaining “the” lookup table:
1 material, many lookup tables?

- Recall: Thinking about textures not as “giftwrap”, but as lookup table for BxDF
- One lookup table per BxDF parameter that needs to be spatially varying
- This example shows: diffuse map, specular map, **normal map**

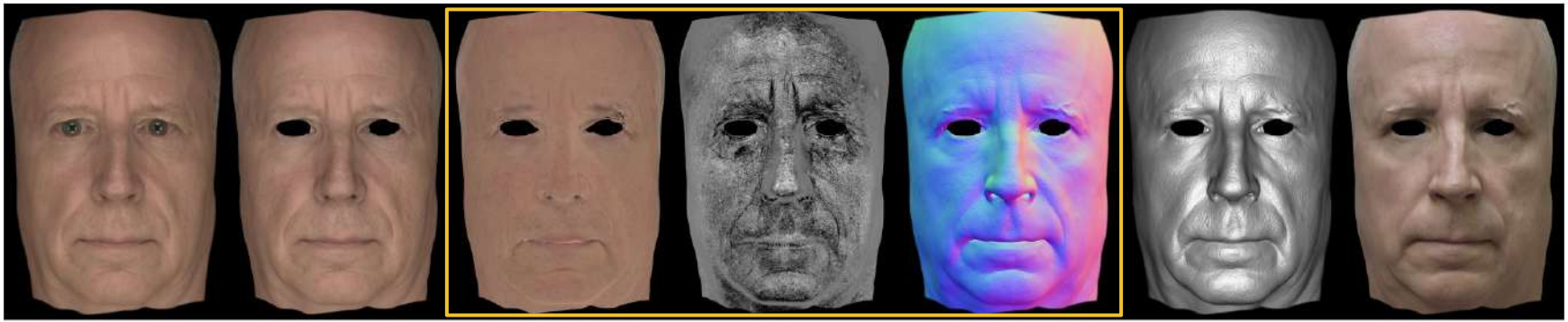
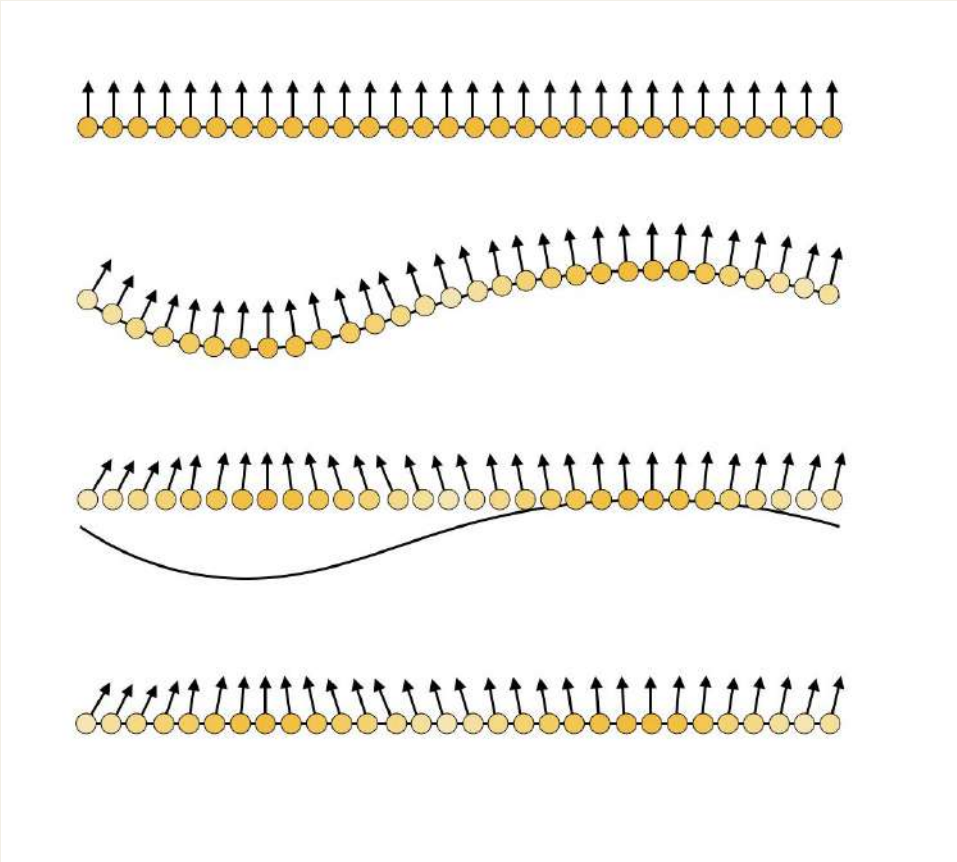


Fig. 1. We present a method to capture complete facial geometry and appearance from a *single* exposure. From left to right: one input image, our matching render, diffuse albedo, specular intensity, normals, high resolution geometry, and a realistic re-render under a different environment map.

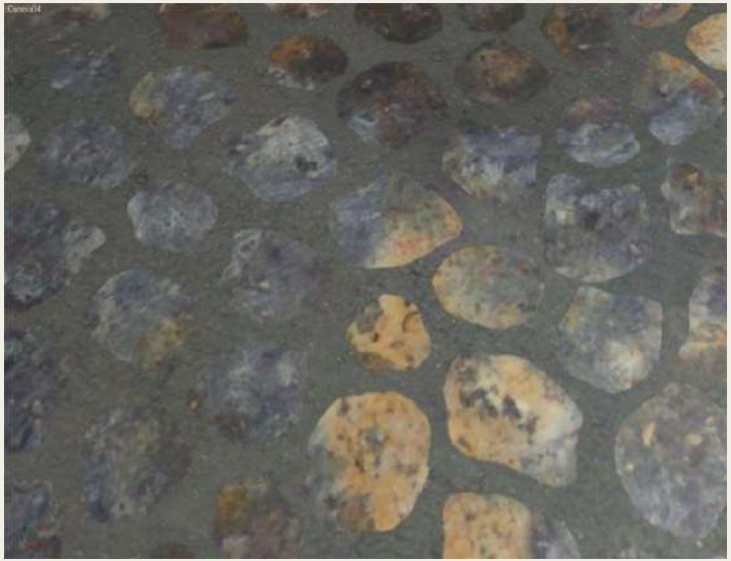
https://studios.disneyresearch.com/2020/08/14/single-shot-high-quality-facial-geometry-and-skin-appearance-capture/

Normal Maps

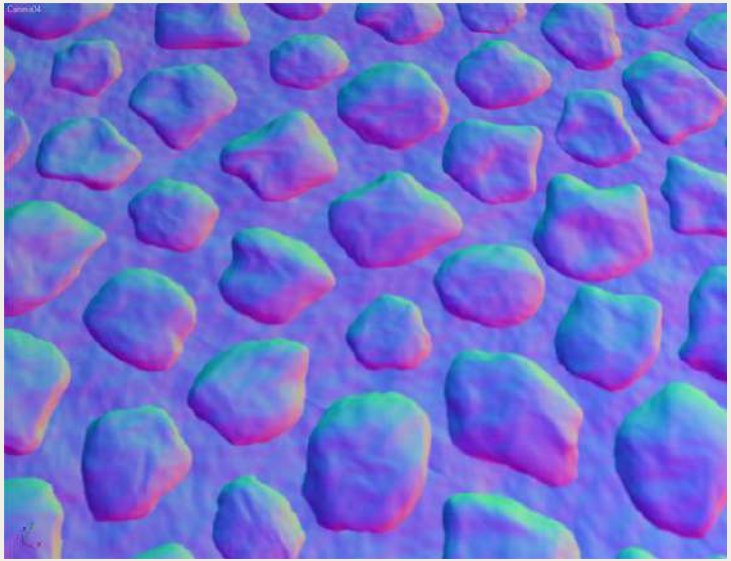


Normal Maps

- Doesn’t change the geometry, simply stores normals (xyz) in an image (rgb)



diffuse map



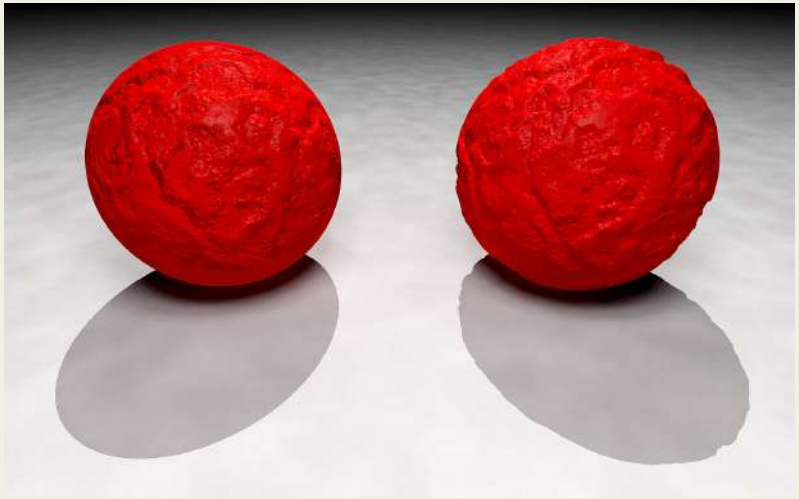
normal map

http://www.bencloward.com/shaders_offset.shtml

Normal Maps

- Doesn’t change the geometry, simply stores normals (xyz) in an image (rgb)
- There are also other techniques such as
  bump mapping (1D height map (greyscale) instead of 3D normals(rgb)) and
  displacement mapping (actual on-the-fly subdivision (Lecture 2) and modification of geometry)

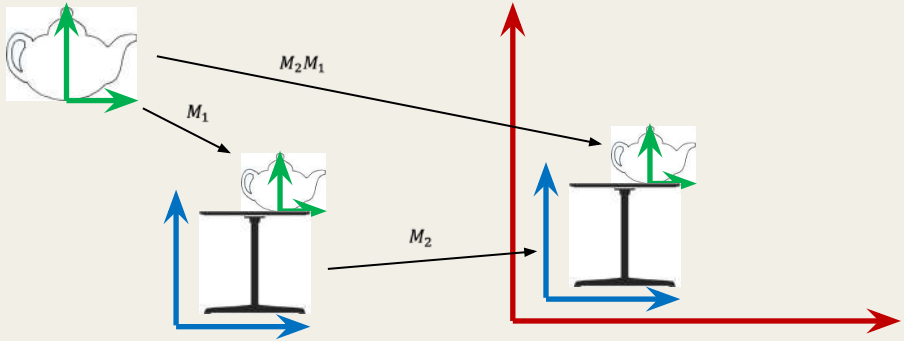
normal mapping on a flat plane (note the variation of specular highlights created by the variation of the normal)



Bump mapping on the left, displacement mapping on the right

1 material, many lookup tables?

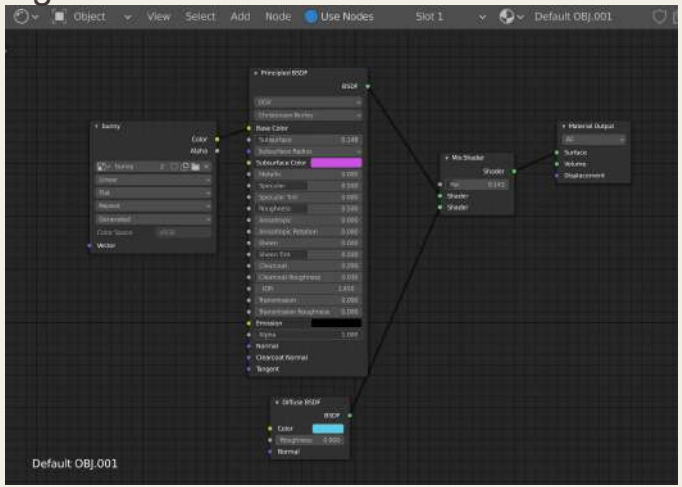
- Often for practical purposes, will still define separate geometry and/or assign separate materials for parts that are too different
  (ex. Belt vs belt buckle)
- Defining hierarchy of objects and using stacked homogenous matrix transforms come in handy (Lecture 4) (again, belt vs buckle)
- Still heavily focus on the diffuse color lookup (“classic” texture mapping)



“1 material, many lookup tables?” in practice

- User interface for many graphics software (Blender, Maya, …) are based on graphs
- Graphical visualization of functions:
  - Has a single output node.
  - Each intermediate node can be thought of as a function
  - Connectors are input/output parameters
  - Computer Graphics: Art supported by a LOT of engineering

- **Blender demo #3! [Homework Content]**



Lecture Outline

- Texture Mapping: UV Coordinates and Creation
- Texture Lookup: Interpolation and Sampling
- Texture Frameworks: One for each BxDF parameter
- Normal Mapping: “Hacking” geometry with textures

- Texture Acquisition: Art, Capture, Procedural
- Environment Mapping and Sky Boxes

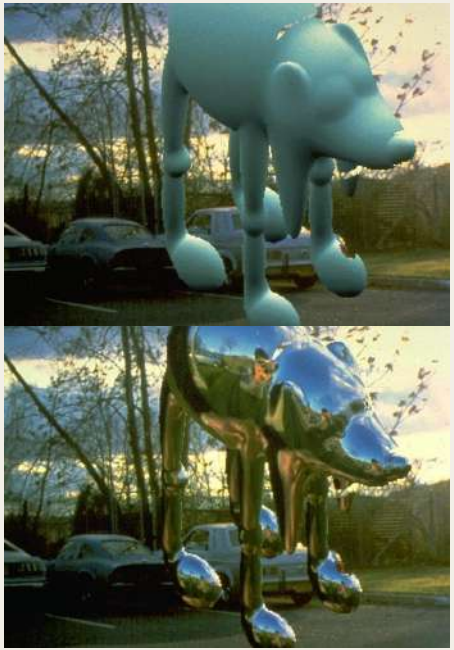
Revisiting the OBJ format

- Lecture 2:
  Store vertices and faces
- Lecture 3:
  Calculate and store per-triangle-vertex normals for shaders
- Now:
  - Store per-triangle-vertex uv coordinates
  - Store materials/textures.
  - Note: Keep this in mind for group projects!
    - materials are not easily portable
    - be careful about relative paths for texture definition

Questions?

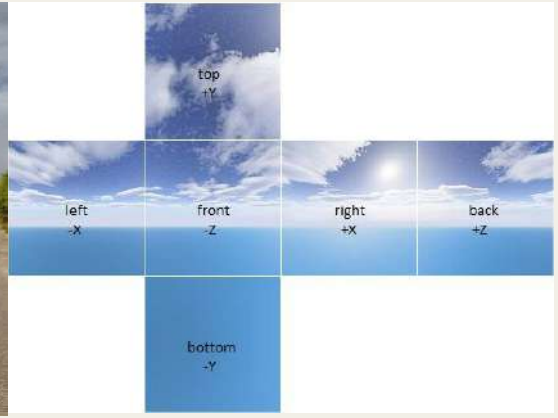
Environment Mapping

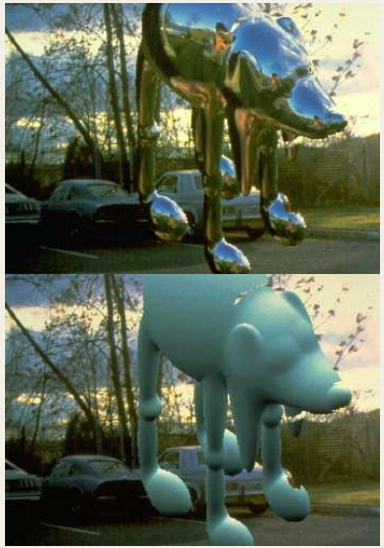
- Recall Lecture 5: Measuring Incoming Light



Environment Mapping

- Obtain realistic global illumination,
  by placing your scene inside a textured cube/sphere,
  and treating the cube/sphere as a **light source**
  (Need Monte Carlo methods to integrate over light source.
  Recall: Path tracing, Distributed ray tracing)





Sky Boxes

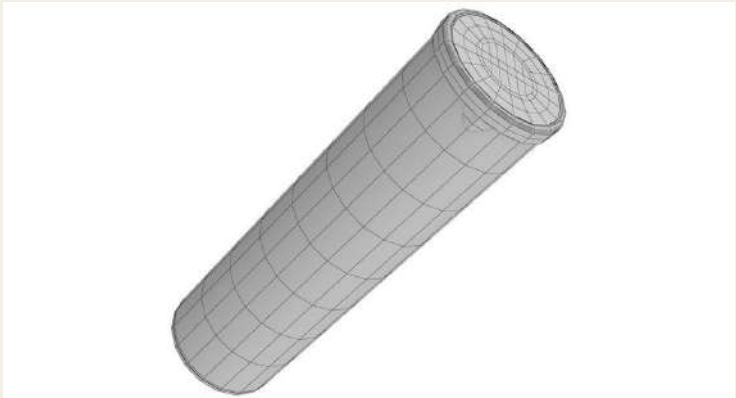
- Avoid explicitly modelling far away/atmospheric objects, by placing your scene inside a textured cube/sphere, and treating the cube/sphere as **geometry** (Need to flip surface normals inwards!)

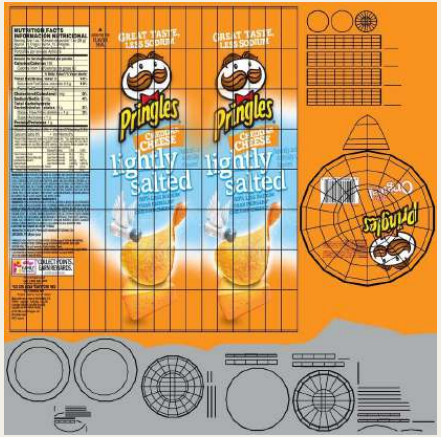
- Very common technique in games
- Thought experiment: Should sky boxes be implemented as ambient, diffuse, or specular lookup?

Recall: UV Coordinates

- Parametrize a surface with 2D UV coordinates,
  that we can use to lookup BxDF colors/parameters from a 2D grid

$$L_o(\omega_o) = \int_{i \in in} BRDF(\omega_i, \omega_o) dE_i(\omega_i)$$





Lecture Outline

- Texture Mapping: UV Coordinates and Creation
- Texture Lookup: Interpolation and Sampling
- Texture Frameworks: One for each BxDF parameter
- Normal Mapping: “Hacking” geometry with textures

- Texture Acquisition: Art, Capture, Procedural
- Environment Mapping and Sky Boxes

Questions?