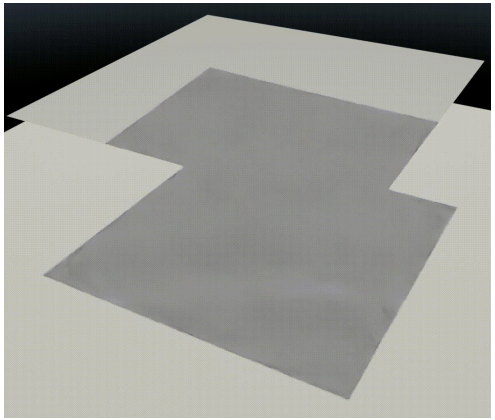
Sampling and Textures

Slides by Sarah Jobalia and Kevin Li, with modifications by Kate Baker

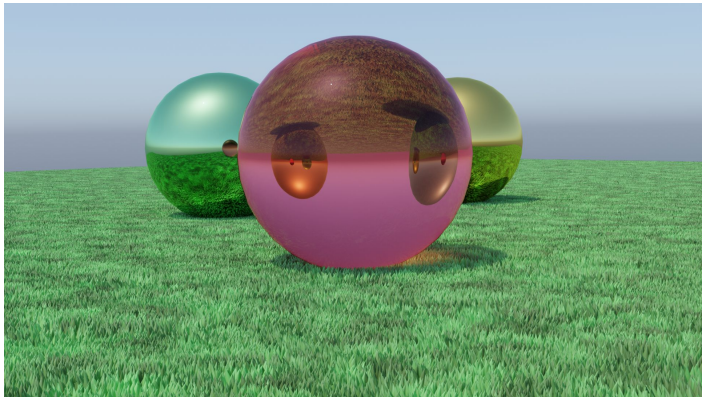
Housekeeping

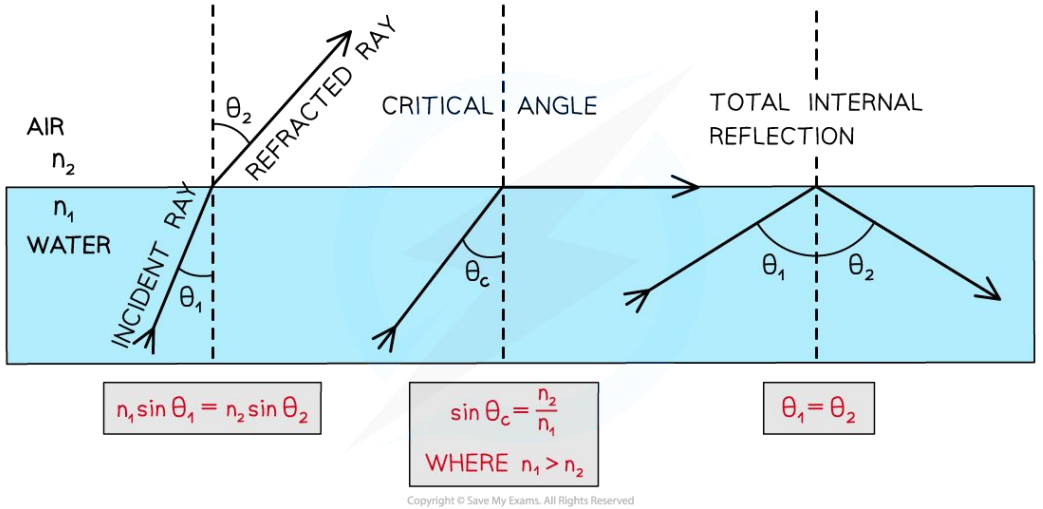
- HW3 due today!
- HW4 coming out soon (due with project proposal)
- Guest speakers! Stay tuned for Ed post today

Last time…



- Recursive raytracing
- Transmission and reflection
  - And their corresponding ray equations
  - Total internal reflection, attenuation, caustics





Lecture Outline

- Wrapping up ray tracing
  - Area lights + sampling, depth of field, color bleeding, global illumination, hybrid methods
- Textures overview
- Texture mapping
  - UV coordinates
- Texture lookup
  - Interpolation + sampling
- Normal mapping

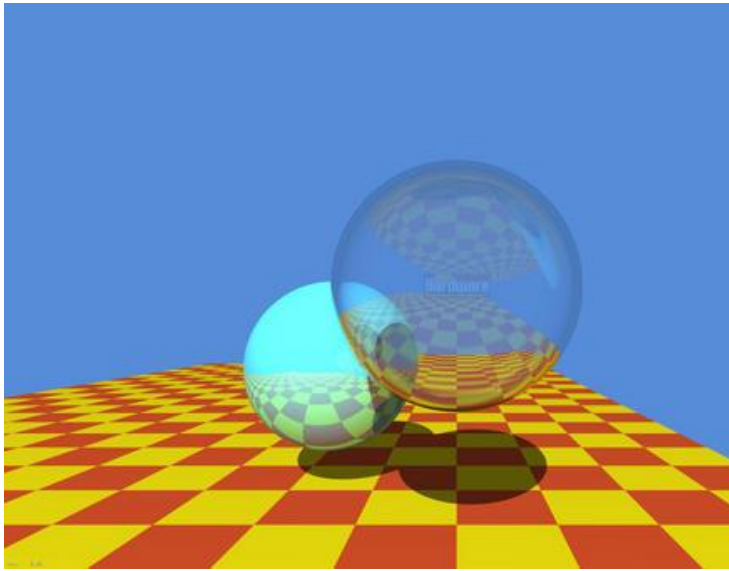
Lecture Outline

- Wrapping up ray tracing
  - Area lights + sampling, depth of field, color bleeding, global illumination, hybrid methods
- Textures overview
- Texture mapping
  - UV coordinates
- Texture lookup
  - Interpolation + sampling
- Normal mapping

Whitted Raytracer

- We’ve covered the very basics of the ray tracing project through the **whitted raytracer**
  - After Turner Whitted (1980)

- What are some things we haven’t covered?

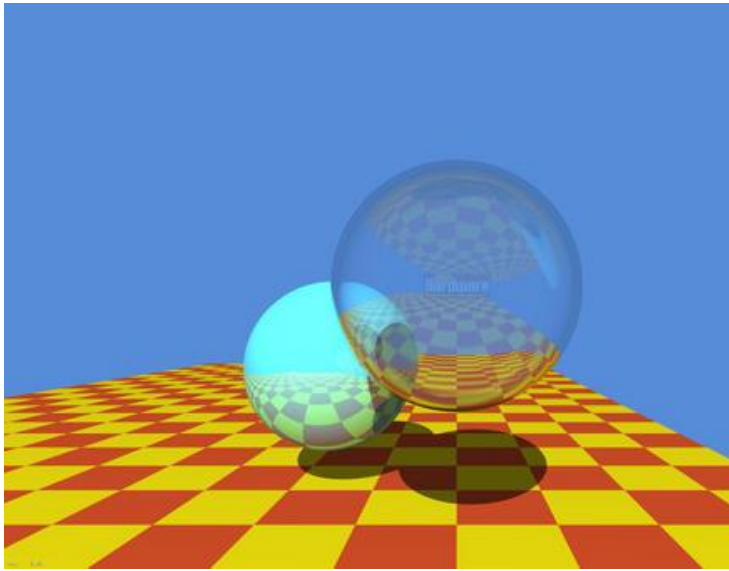


Turner Whitted’s iconic 1980 rendering of a sphere and checkerboard using his ray tracing algorithm. It was unprecedented for the time because of the reflections, shadows, and refraction seen in the spheres.
https://sites.williams.edu/scientephic/news/turner-whitted-and-virtual-realityfrom-promise-to-practical/

Whitted Raytracer

- We’ve covered the very basics of the ray tracing project through the **whitted raytracer**
  - After Turner Whitted (1980)

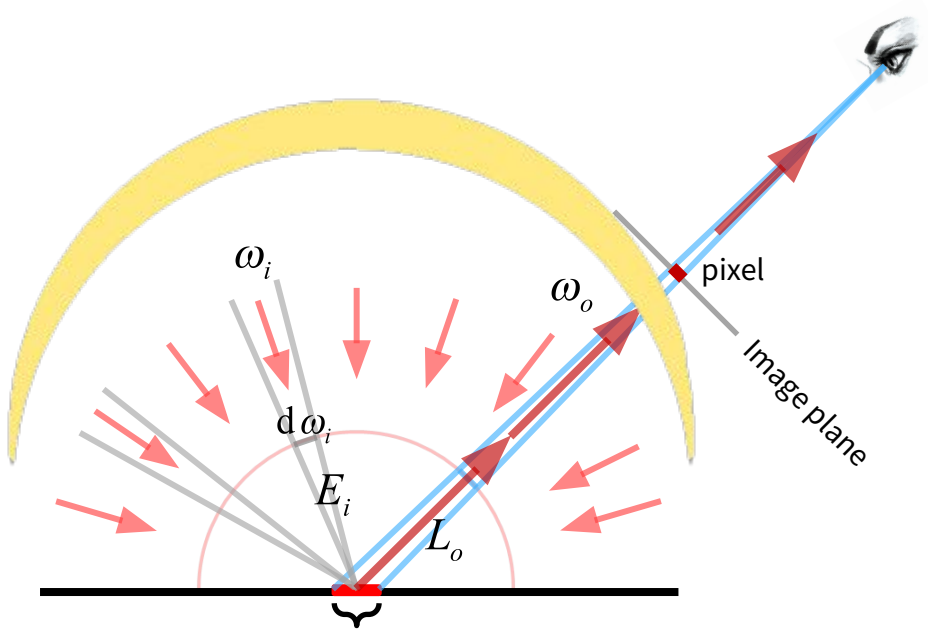
- What are some things we haven’t covered?
  - Area lights
  - Depth of field
  - Color bleeding



Turner Whitted’s iconic 1980 rendering of a sphere and checkerboard using his ray tracing algorithm. It was unprecedented for the time because of the reflections, shadows, and refraction seen in the spheres.
https://sites.williams.edu/scientephic/news/turner-whitted-and-virtual-realityfrom-promise-to-practical/

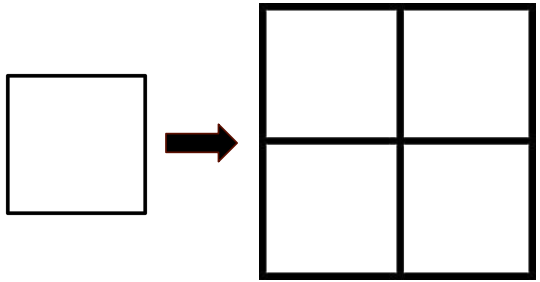
Area Lights

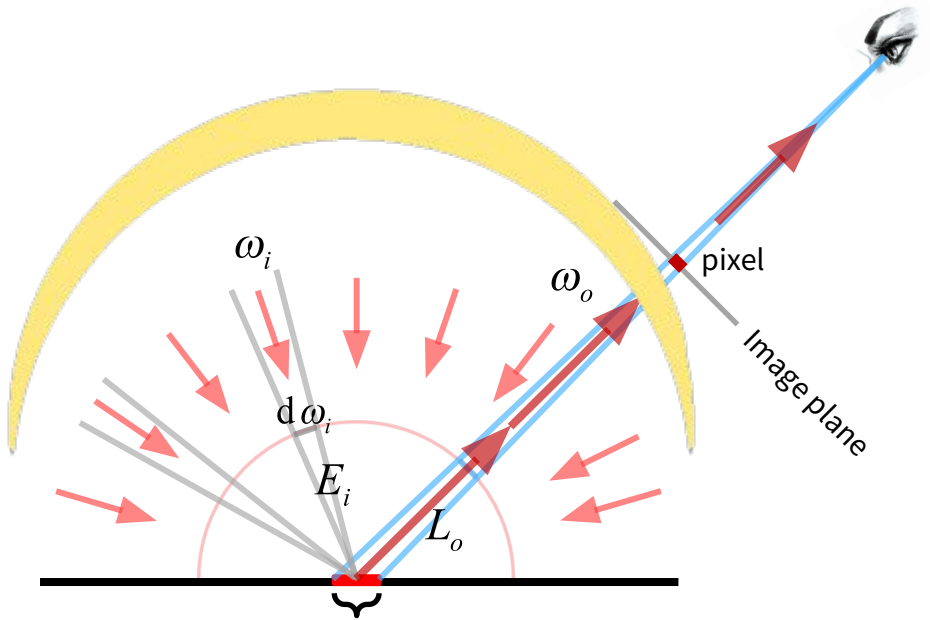
- An **area light** is comparable to a collection of point lights
- We could send out a ray for every point that makes up the area light…



Area Lights

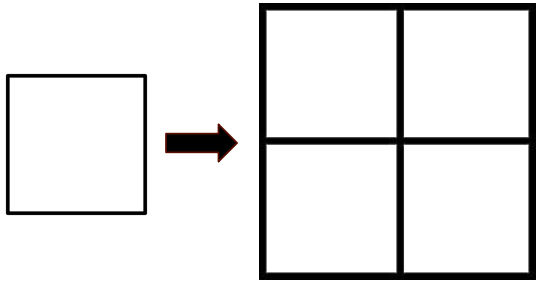
- An **area light** is comparable to a collection of point lights
- We could send out a ray for every point that makes up the area light…
- But then what if we double the length of the area light?



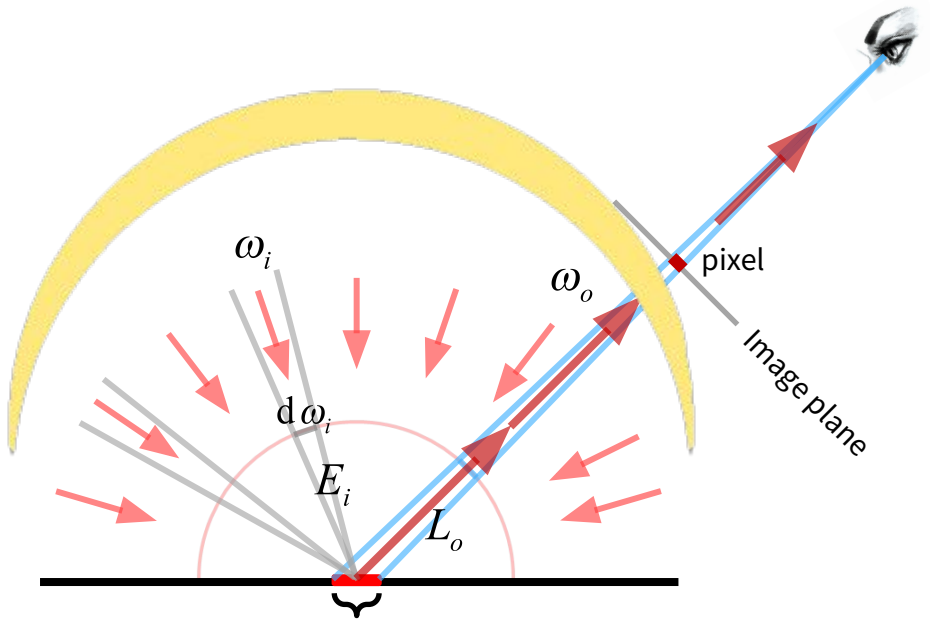


Area Lights

- An **area light** is comparable to a collection of point lights
- We could send out a ray for every point that makes up the area light…
- But then what if we double the length of the area light?

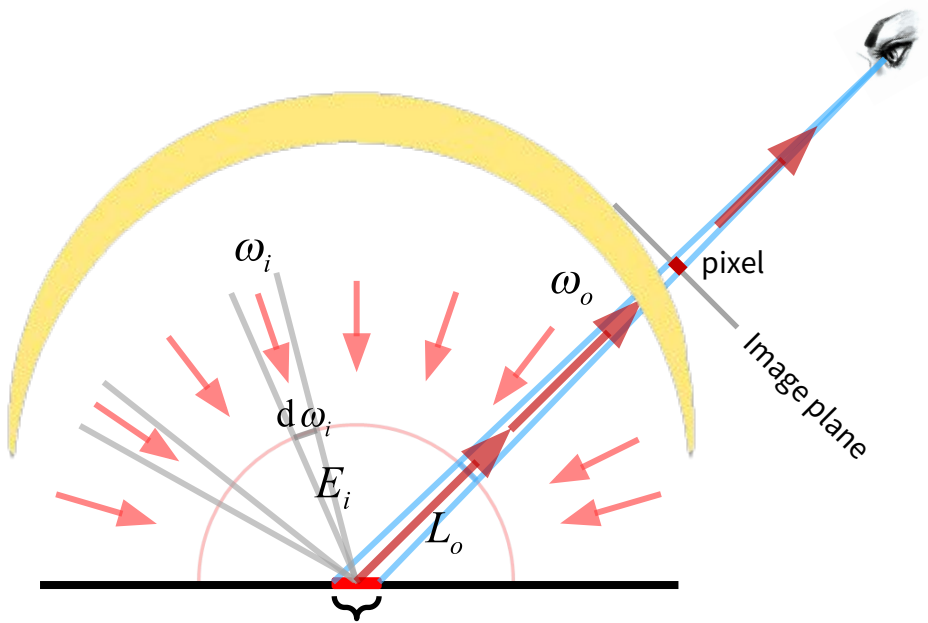


- 4x as many points now :(
  - **Curse of dimensionality**



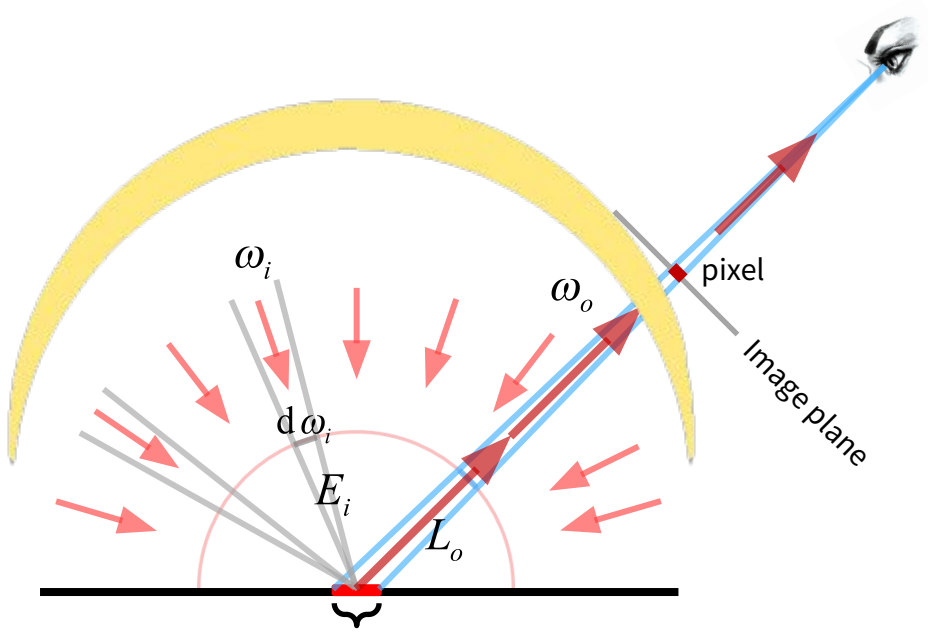
Area Lights

- An **area light** is comparable to a collection of point lights
- What should we do instead?



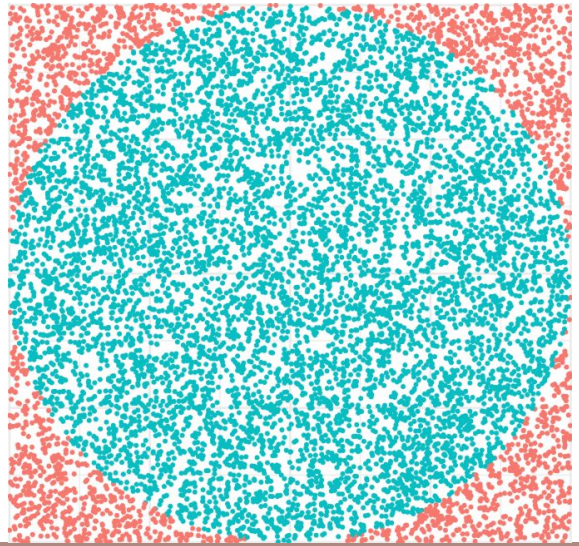
Area Lights

- An **area light** is comparable to a collection of point lights
- What should we do instead?
  - We can **random sample** rays on the light's surface



Sampling Techniques: Monte Carlo

- Relying on **repeated** random sampling to get results
- Example: estimating π
  - Consider square of size 2x2, then put a circle of radius 1 inside
  - Throw darts at the square and color those that hit inside the circle blue, outside red

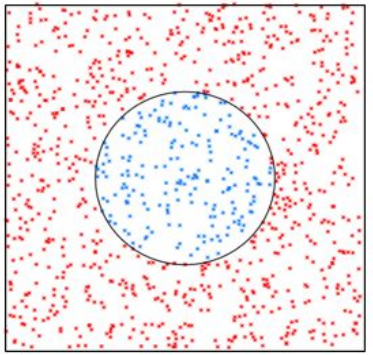


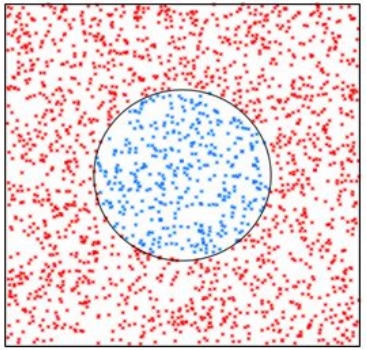
$$\pi \approx 4\frac{\#blue}{\#red + \#blue}$$

$$\pi \approx 3.1440$$

Sampling Techniques: Monte Carlo

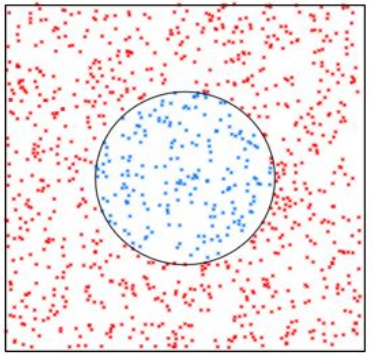
- Relying on **repeated** random sampling to get results
- Slow convergence

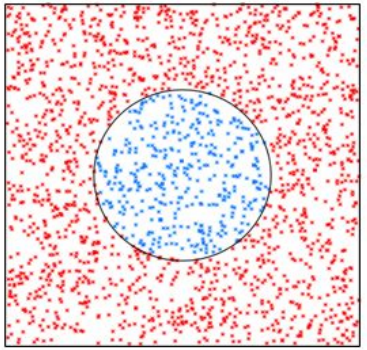




Sampling Techniques: Monte Carlo

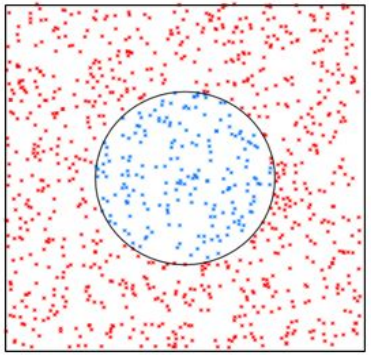
- Relying on **repeated** random sampling to get results
- Slow convergence
- **Independent** of the number of dimensions!

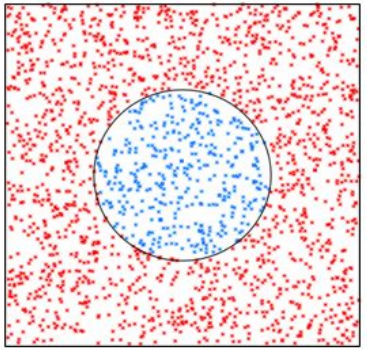




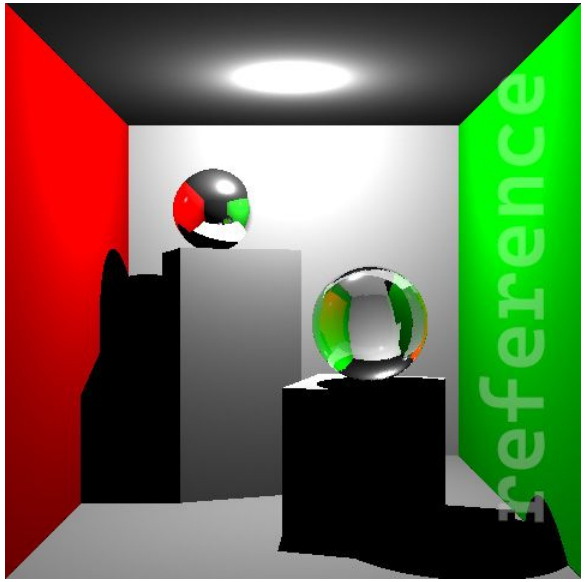
Sampling Techniques: Monte Carlo

- Relying on **repeated** random sampling to get results
- Slow convergence
- **Independent** of the number of dimensions!
- Example: Disk-shaped area light
  - Randomly sample 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

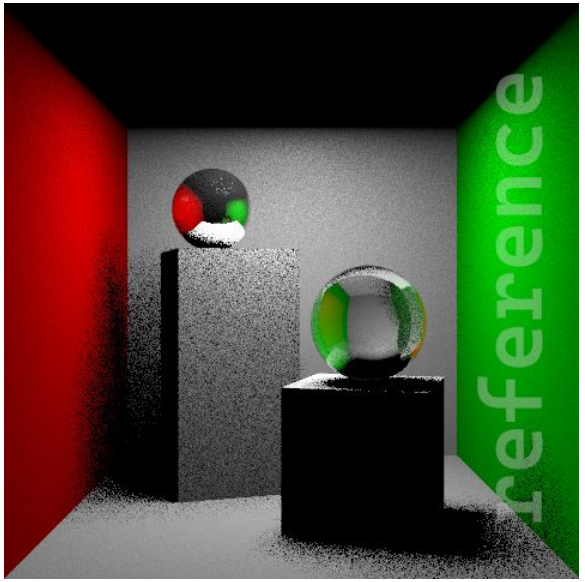




Sampling Area Lights

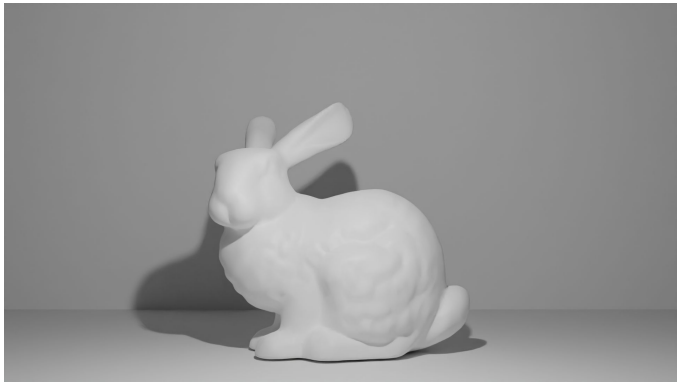


ray tracing with just point lights

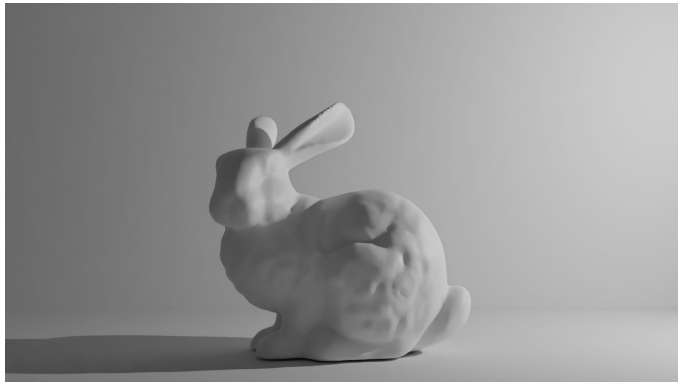


ray tracing with area lights (4 samples)

Different Types of Lights



Point Light

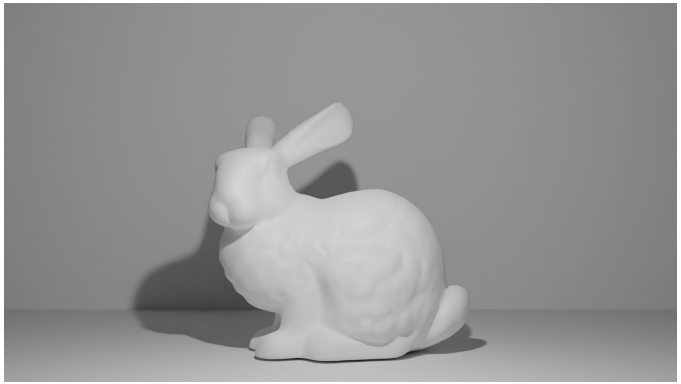


Area Light

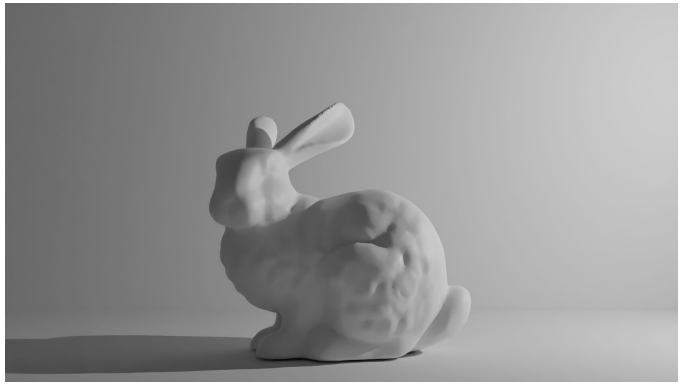
Directional (Sun) Light

Spotlight

Different Types of Lights



Point Light

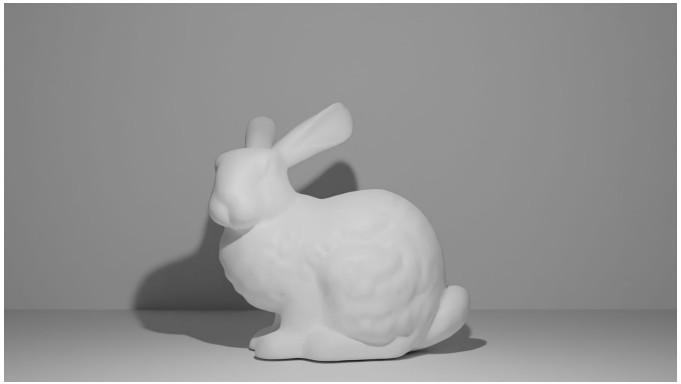


Area Light

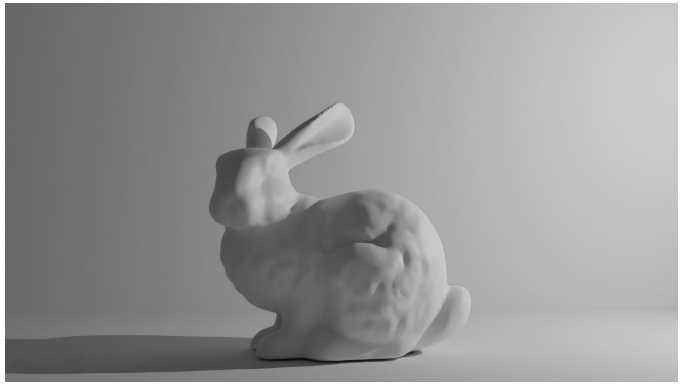
Directional (Sun) Light

Spotlight

Different Types of Lights



Point Light

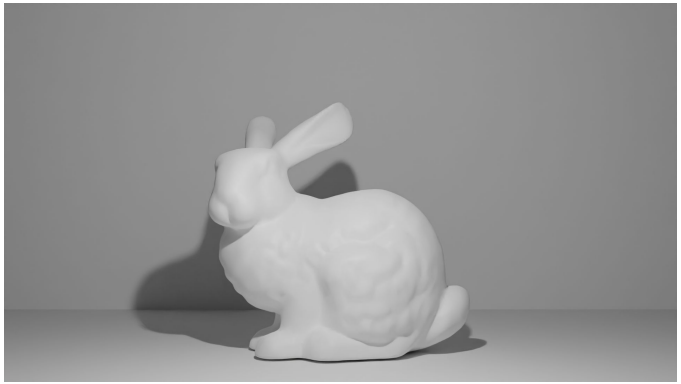


Area Light

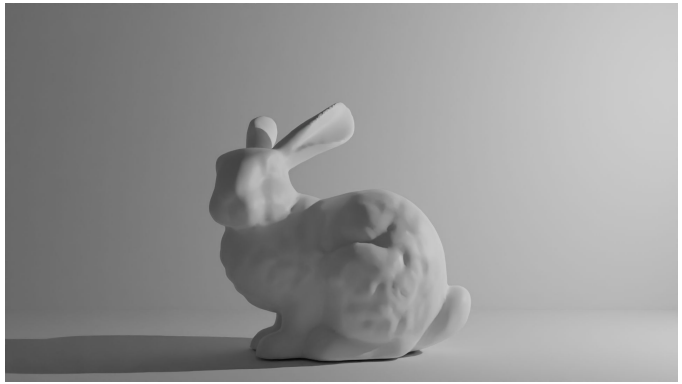
Directional (Sun) Light

Spotlight

Different Types of Lights



Point Light



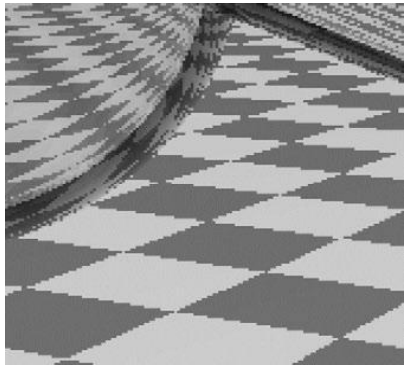
Area Light

Directional (Sun) Light

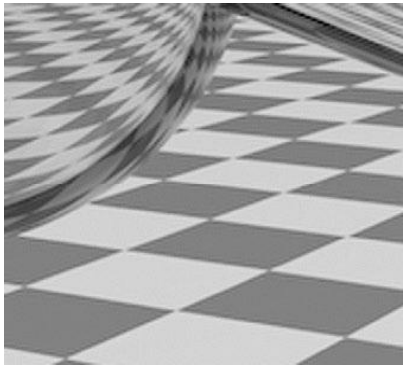
Spotlight

Sampling Camera Rays: Anti-Aliasing

- Sending one ray through each pixel of our film plane
  - Leads to **aliasing**: pixelated edges in our render



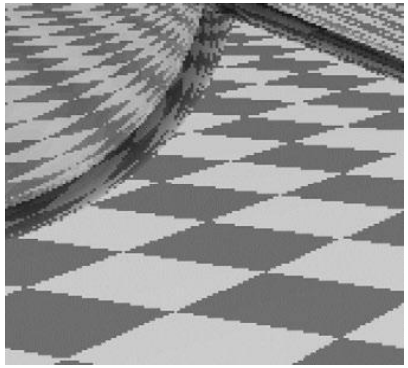
aliased



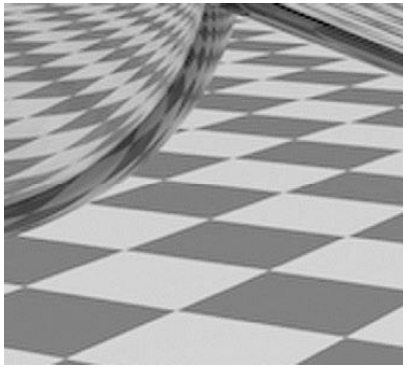
anti-aliased

Sampling Camera Rays: Anti-Aliasing

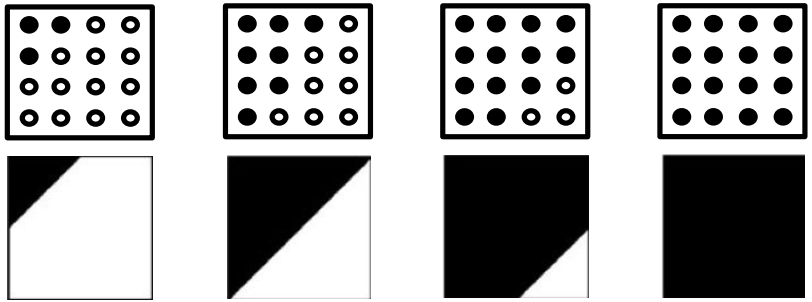
- Sending one ray through each pixel of our film plane
  - Leads to **aliasing**: pixelated edges in our render
- Solution: make the pixel finer grain → sample camera rays through random points in the pixel → average the render results



aliased

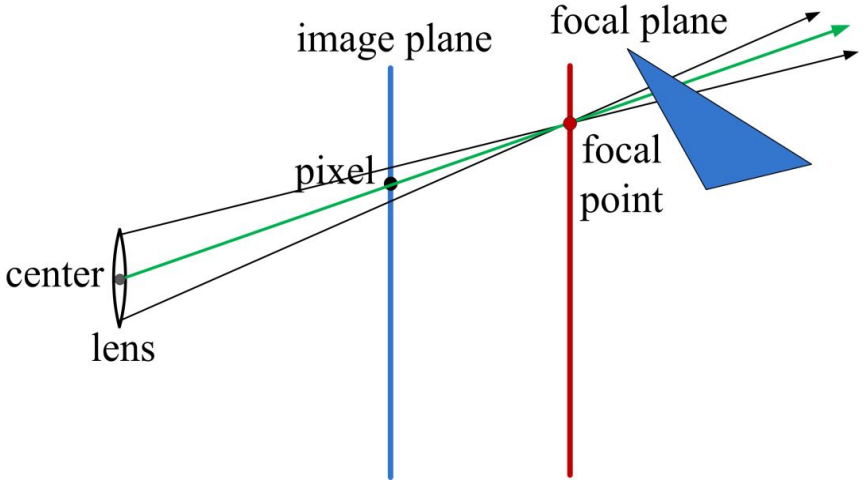


anti-aliased



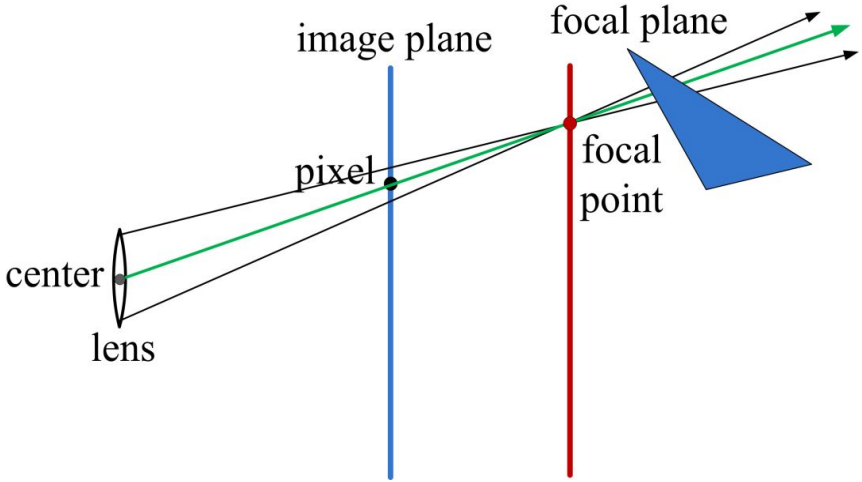
Sampling Camera Rays: Depth of Field

- Randomly sample camera rays around the circumference of the 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)



Sampling Camera Rays: Depth of Field

- Randomly sample camera rays around the circumference of the 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)
- For a single pixel:
  - Sample many points across the lens aperture
  - For each sample, trace a ray that starts there and passes through focal point
  - Average colors together



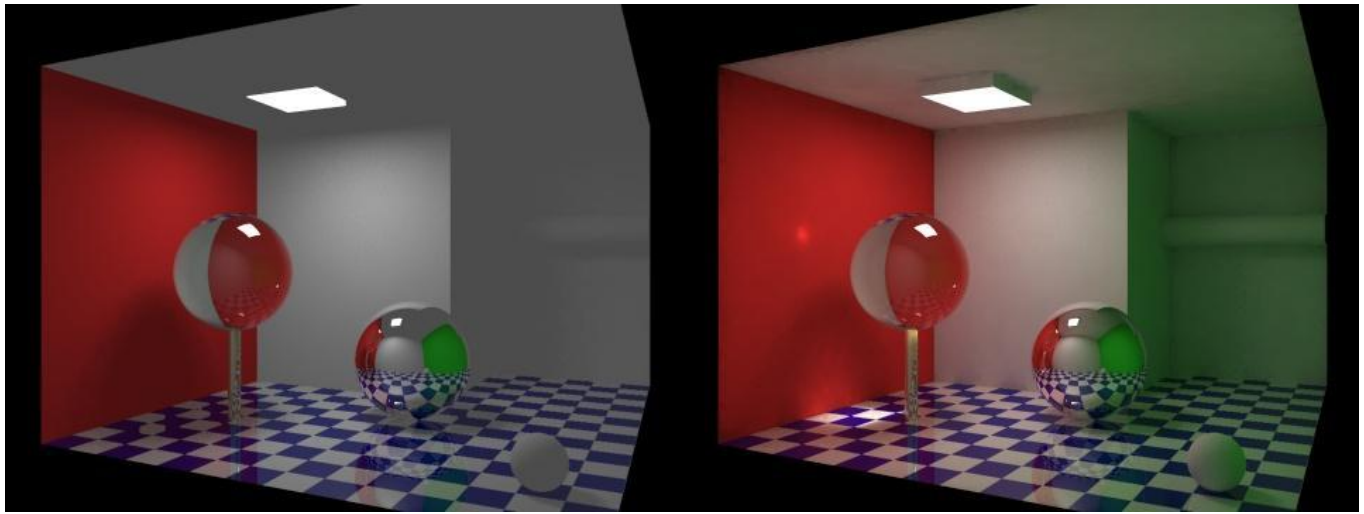
Whitted → Distributed Raytracer

- Distributed = stochastic ray tracing
- Using 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.

Color Bleeding (Again)

- In the real world, light can come from other things besides 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

Global Illumination

- 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 hit on the way should contribute their color for color bleeding

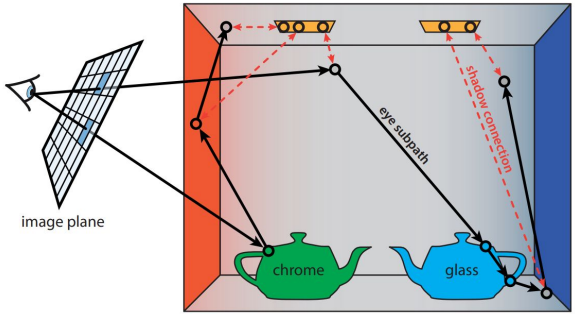
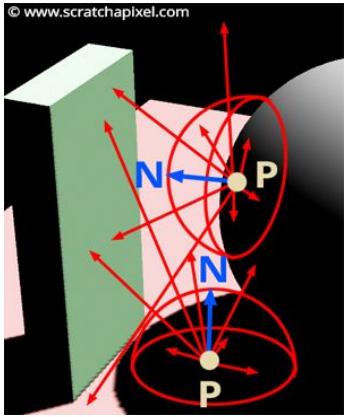


Figure 3.1: An illustration of tracing paths from the eye to the light sources in a Cornell box scene with two teapots.

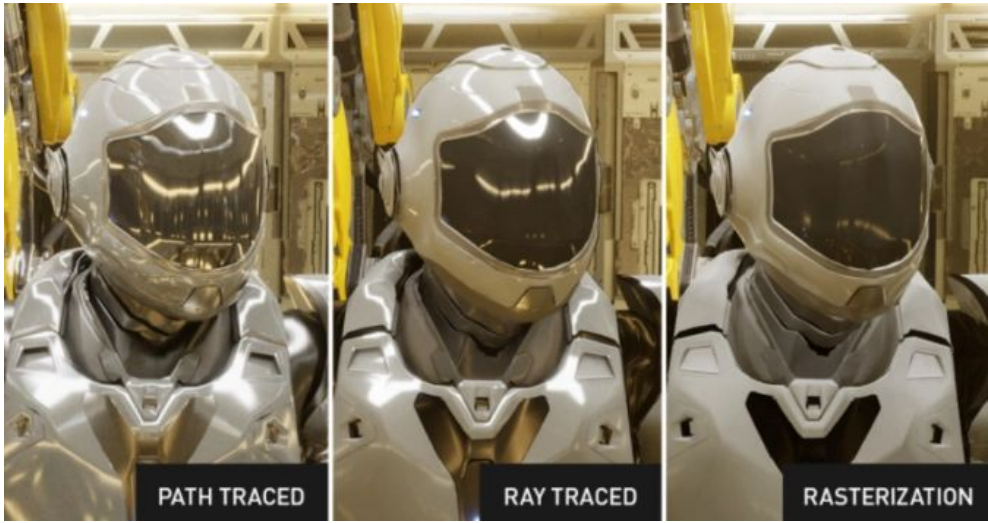
https://graphics.pixar.com/library/PathTracedMovies/paper.pdf



Global Illumination: Path Tracing

- The most realistic result can be inefficient – not all scattered rays may lead back to a light within a reasonable number of bounces

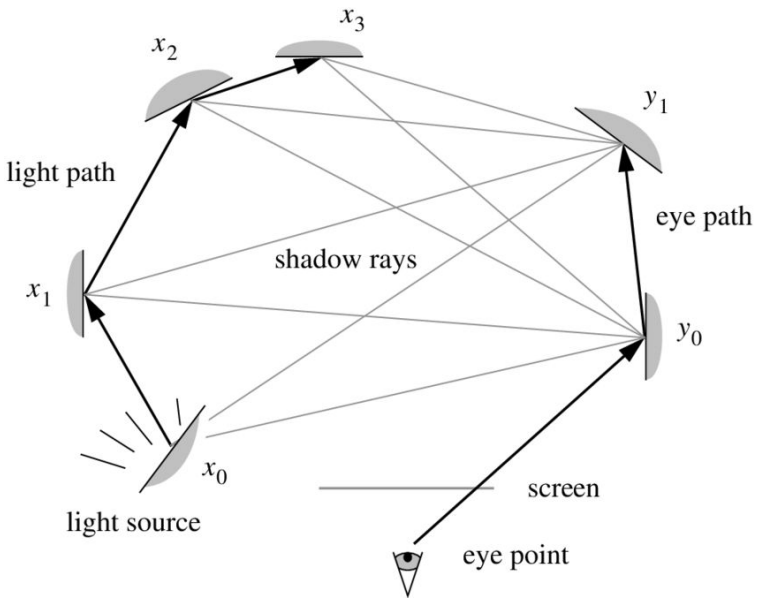
- Process often combined with **smarter (importance) sampling**
- Or approximated methods like bidirectional ray tracing and photon mapping…



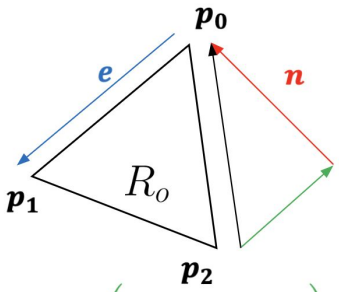
https://blogs.nvidia.com/blog/2022/03/23/what-is-path-tracing/

Global Illumination: Hybrid Methods

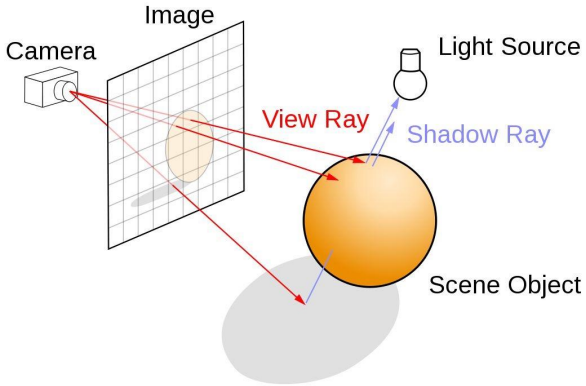
- **Bidirectional ray tracing**
  - Send rays from both the camera and light sources

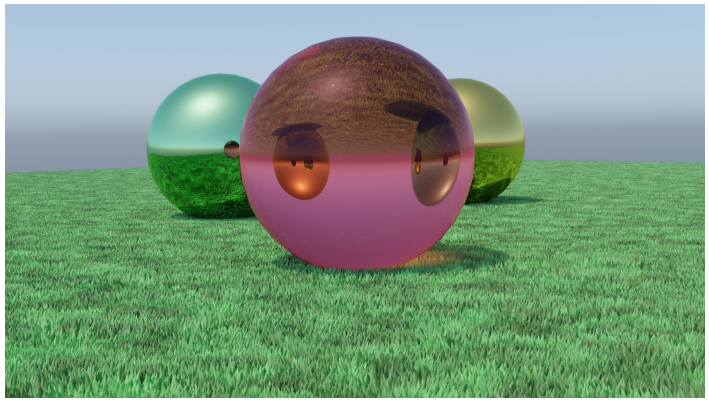


Ray Tracing!!



$$\left((p_0 - p_2) \cdot \frac{e}{\|e\|}\right)\frac{e}{\|e\|}$$



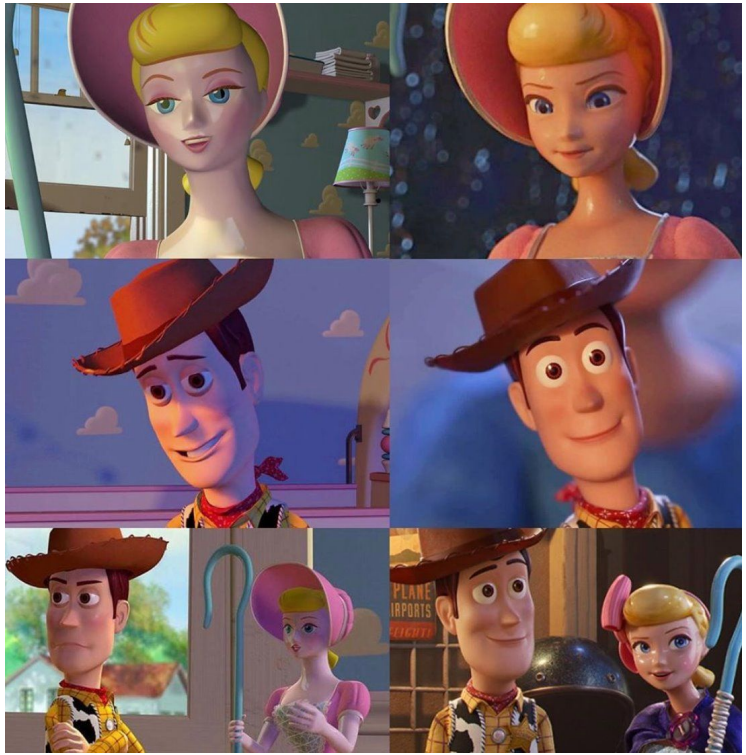


Lecture Outline

- Wrapping up ray tracing
  - Area lights + sampling, depth of field, color bleeding, global illumination, hybrid methods
- Textures overview
- Texture mapping
  - UV coordinates
- Texture lookup
  - Interpolation + sampling
- Normal mapping

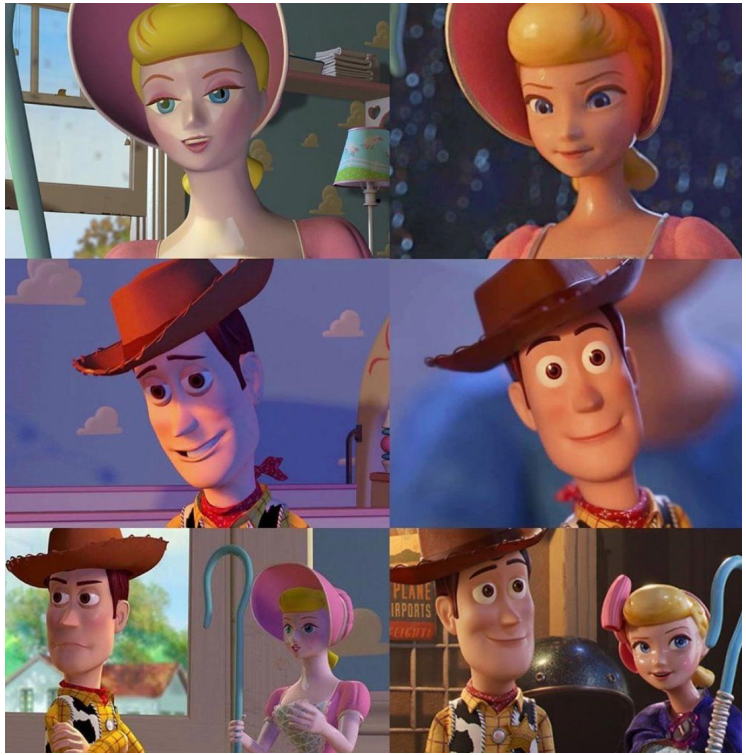
Choosing a Texturing Technique

- **Textures**: associate different colors to a single object
- Stylization is often a choice due to complexity constraints



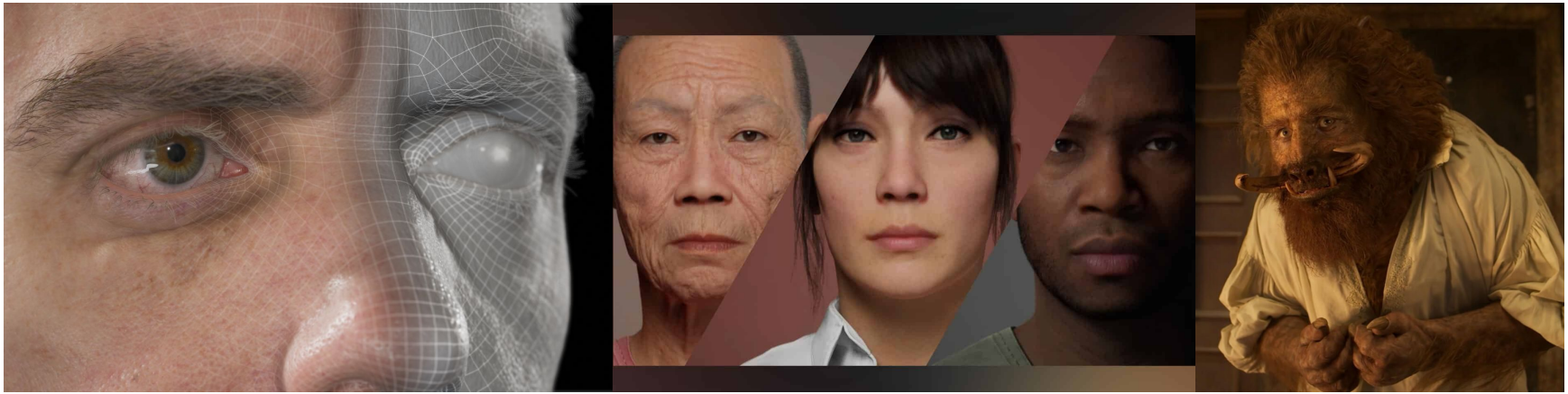
Choosing a Texturing Technique

- **Textures**: associate different colors to a single object
- Stylization is often a choice due to complexity constraints
  - Realistic environments aren't always the goal
    - Risks the uncanny valley
  - E.g. Toy Story – it's aged well!
    - Rasterization, not raytraced, smooth/plastic toys



Choosing a Texturing Technique

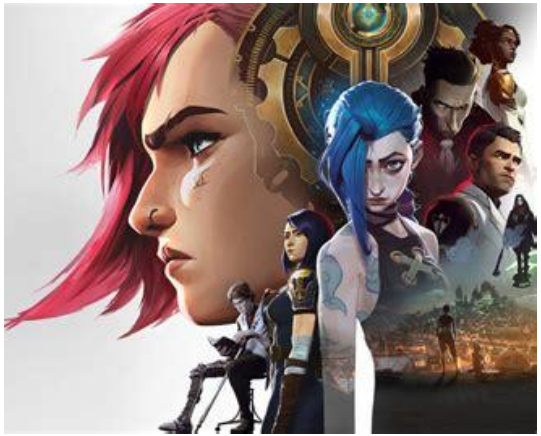
- Risks the **uncanny valley**
  - It takes a lot of effort to make humanoid appearance and motion look “not creepy” / “not game like”
  - Check out these links!



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/

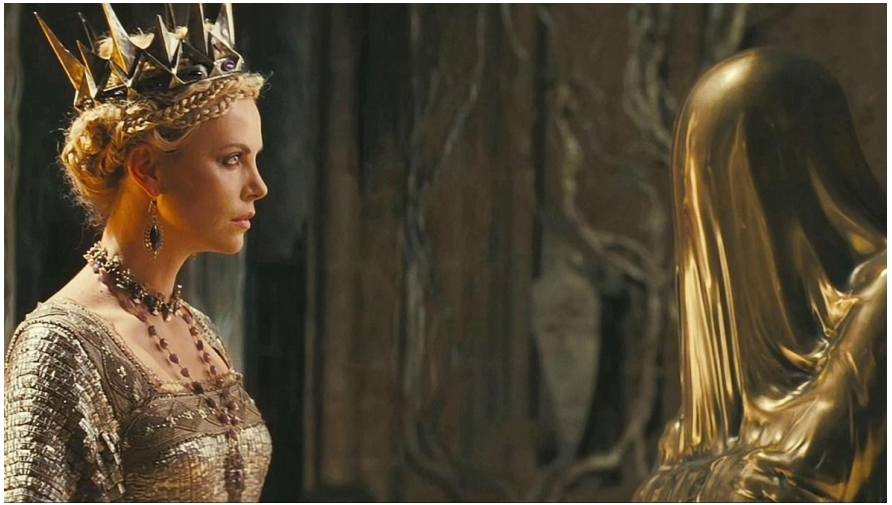
Choosing a Texturing Technique

- These days, texturing is more of an aesthetic choice rather than a constraints choice
- We still choose to make things look less realistic
  - E.g. Spiderverse: extra effort to make CGI look hand-drawn and non-3D



Choosing a Texturing Technique

- Realism is often based on capturing real-world materials, geometry, motion, and/or lighting
  - E.g. Snow White and the Huntsman: Witness Camera to capture lighting to map onto a mirror creature



Choosing a Texturing Technique

- Can see the reflection on the CGI creature!
  - Many fundamentals from previous lectures will continue to appear in different practical considerations!

Yan (Mia) Miao

Bihan Liu and Yixin Liu

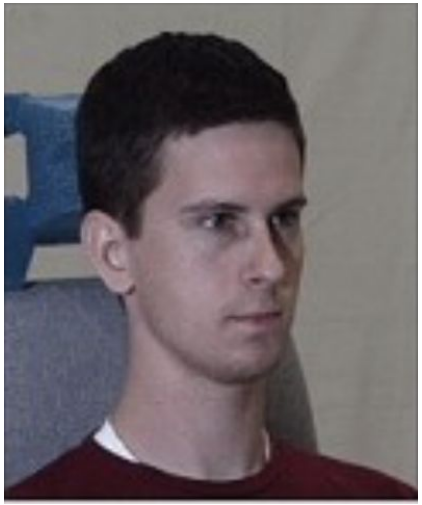
Enci Liu

Lecture Outline

- Wrapping up ray tracing
  - Area lights + sampling, depth of field, color bleeding, global illumination, hybrid methods
- Textures overview
- Texture mapping
  - UV coordinates
- Texture lookup
  - Interpolation + sampling
- Normal mapping

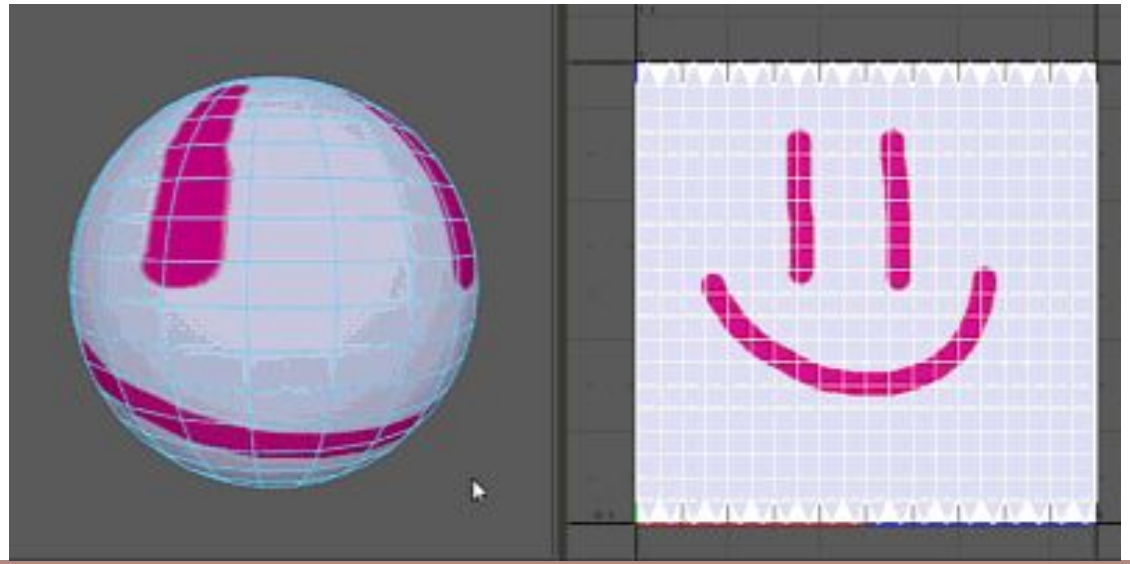
Textures

- We want more complex appearance beyond solid color
- Technique solution: a “giftwrap” approach



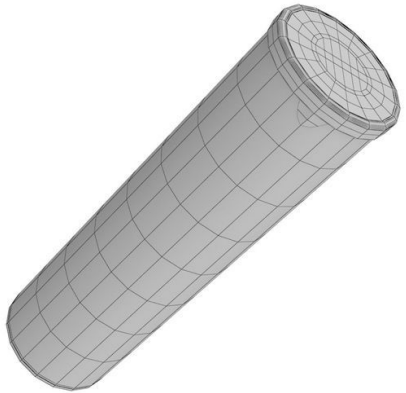
Textures

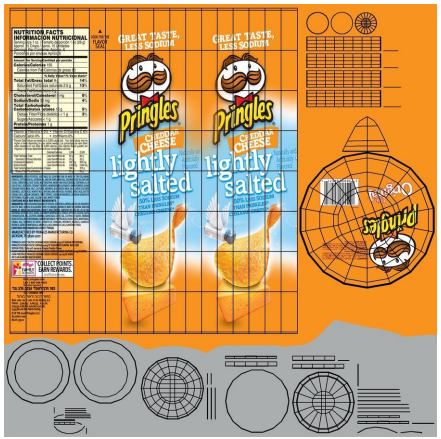
- “Giftwrap” approach: parameterize geometric surfaces with a **2D coordinate system** so that we can paste images (textures) onto geometry



Textures

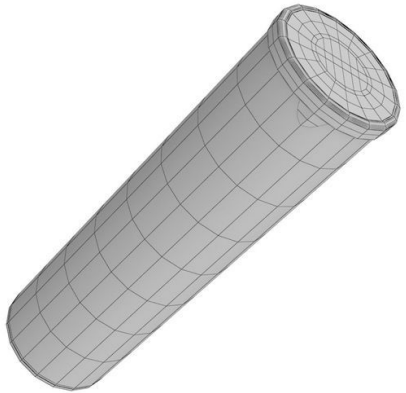
- “Giftwrap” approach: parameterize geometric surfaces with a **2D coordinate system** so that we can paste images (textures) onto geometry

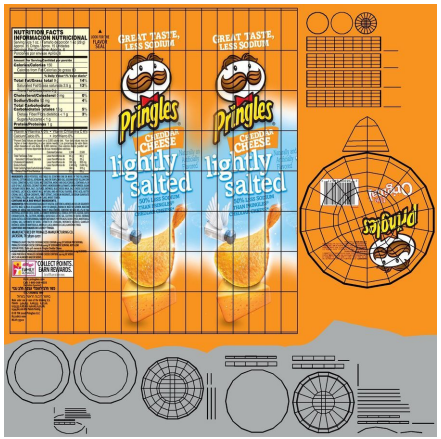




Textures

- Recall: rasterization (splatting) → raytracing (point-based)
- Creating many BxDFs (BRDF/BTDF/BSSRDF) to achieve separate looks is **inefficient**.
- Instead: have a single **spatially varying** BxDF

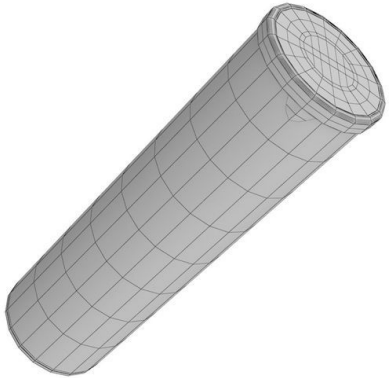




Textures

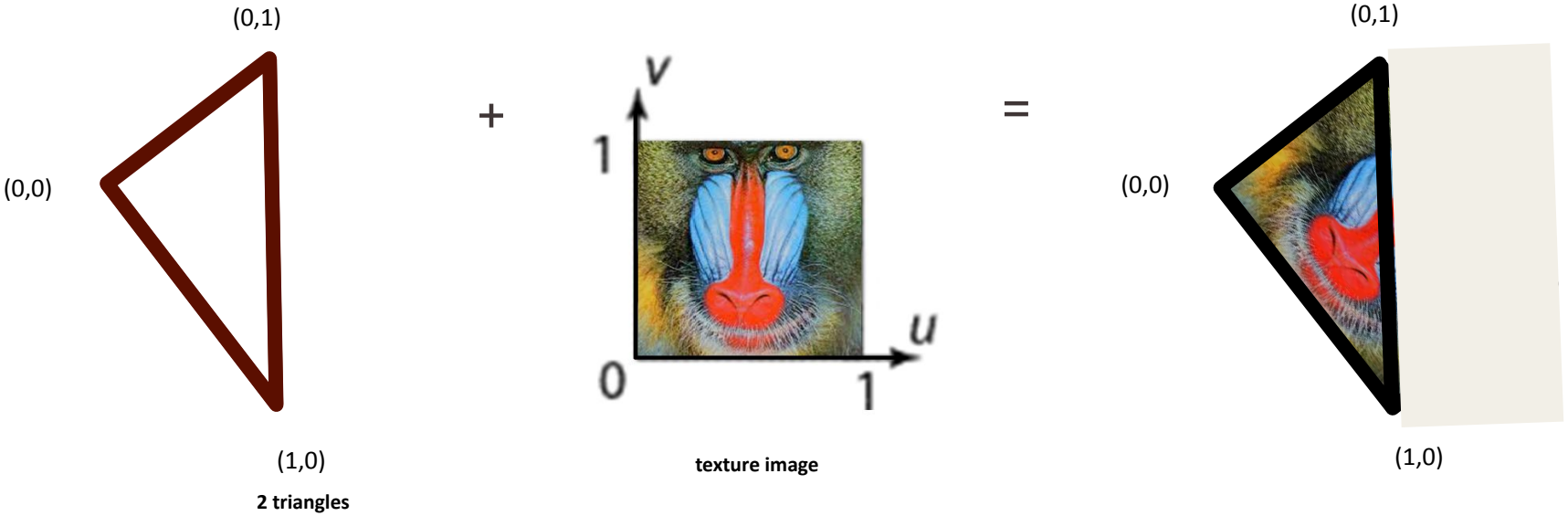
- We can have a more general “rendering equation”-based way to frame this
- Define a 2D function on the surface
  - Function outputs coordinates that we can use to lookup BxDF colors/parameters from a 2D grid

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



UV Coordinates

- A more formal approach
- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties



UV Coordinates: OBJ File

- A more formal approach
- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties

```
v 0.123 0.234 0.345
```

A vertex in 3D space (an XYZ position in the mesh's geometry)

```
vt 0.500 1
```

```
f v1 v2 v3
```

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

UV Coordinates: OBJ File

- A more formal approach
- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties

```
v 0.123 0.234 0.345
```

A vertex in 3D space (an XYZ position in the mesh's geometry)

```
vt 0.500 1
```

A vertex texture (vt) coordinate and usually stored in a separate list

```
f v1 v2 v3
```

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

UV Coordinates: OBJ File

- A more formal approach
- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties

```
v 0.123 0.234 0.345
```

A vertex in 3D space (an XYZ position in the mesh's geometry)

```
vt 0.500 1
```

A vertex texture (vt) coordinate and usually stored in a separate list

```
f v1 v2 v3
```

A face defining a triangle

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

UV Coordinates: OBJ File

- A more formal approach
- Assign 2D (UV) coordinates to each point on the surface (XYZ)
- Have a lookup table from UV space to material properties

```
v 0.123 0.234 0.345
```

A vertex in 3D space (an XYZ position in the mesh's geometry)

```
vt 0.500 1
```

A vertex texture (vt) coordinate and usually stored in a separate list

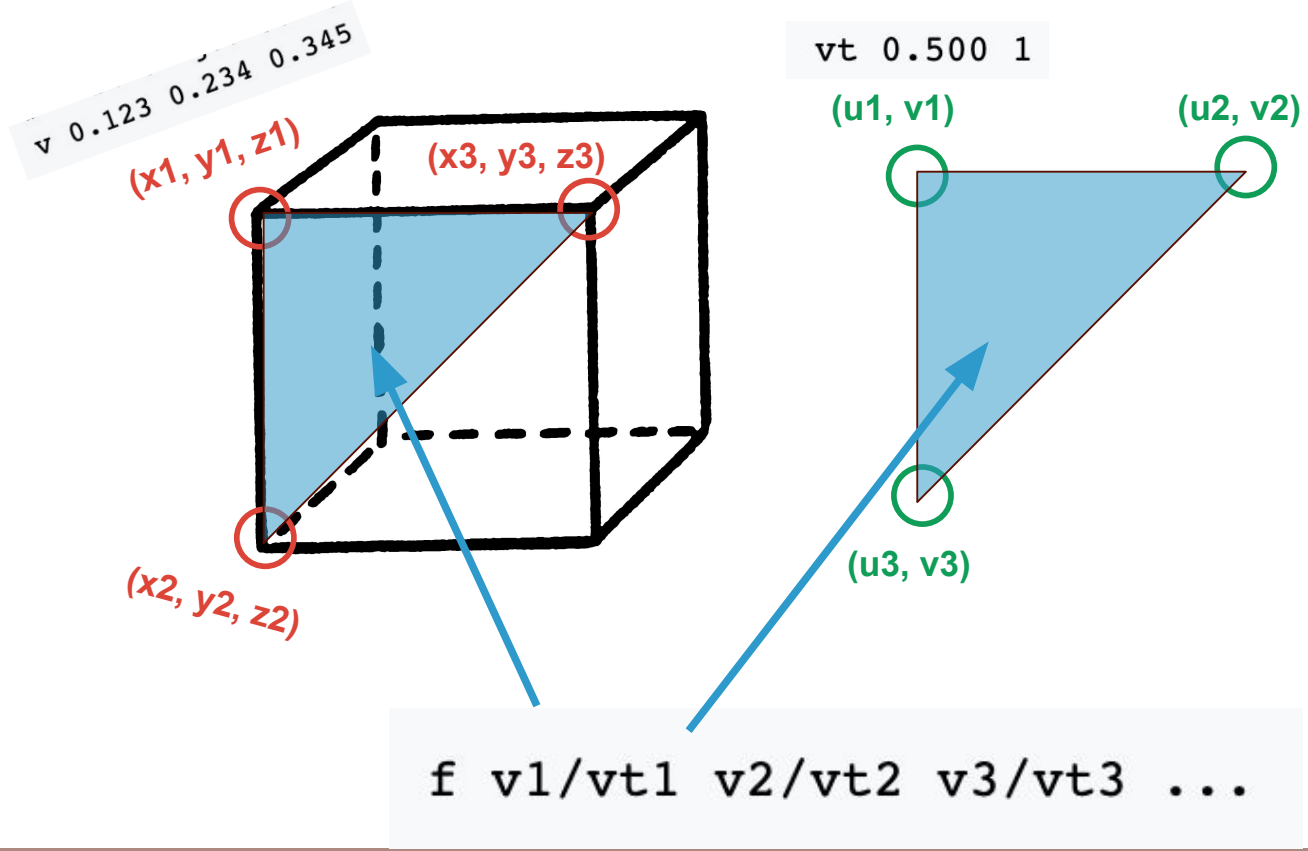
```
f v1 v2 v3
```

A face defining a triangle

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

^ The same face definition where each vertex is paired with a texture coordinate

UV Coordinates: OBJ File



UV Coordinates: Assignment

- Can construct analytical functions for implicit geometry (sphere, cylinder, plane)
- Many options even for a single type of geometry

3.3 Tessellating a Sphere: Vertices

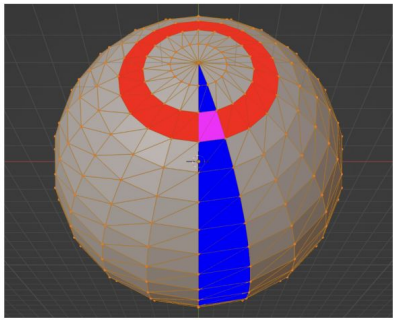
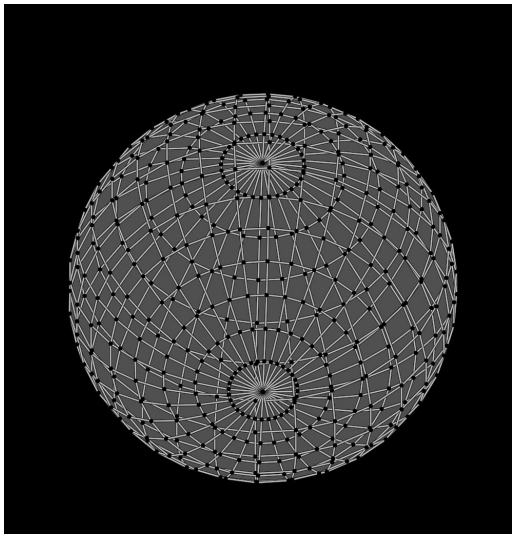
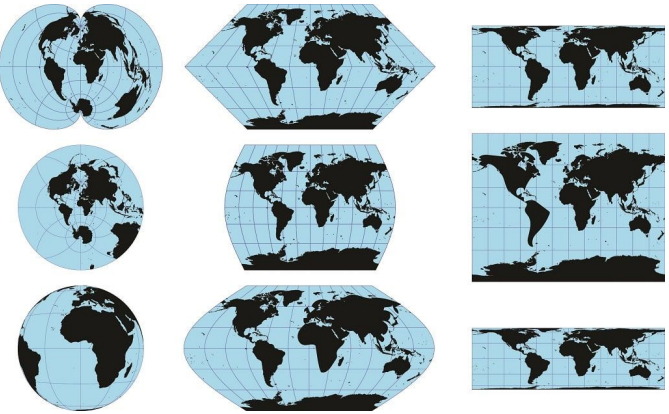


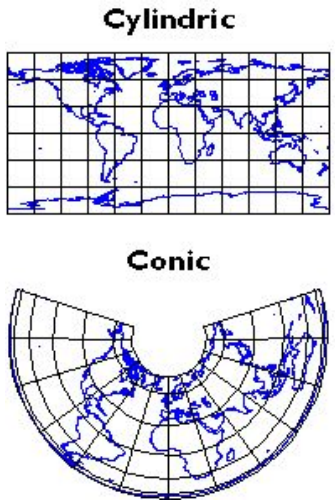
Figure 1: We can divide the surface of a sphere up into latitudinal faces called stacks (in blue) and longitudinal faces called sectors (in red).

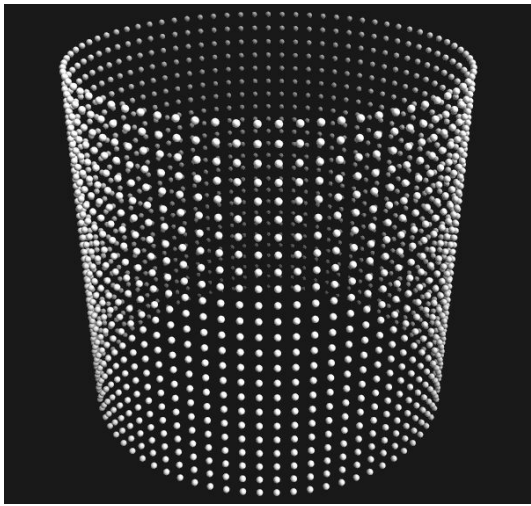


UV Coordinates: Assignment

- Can also define **per-polygon** for explicit geometry (polygon meshes)
- Now, we have a list of vertex positions, a list of UV coordinates, and a list of indices into the vertex + UV list

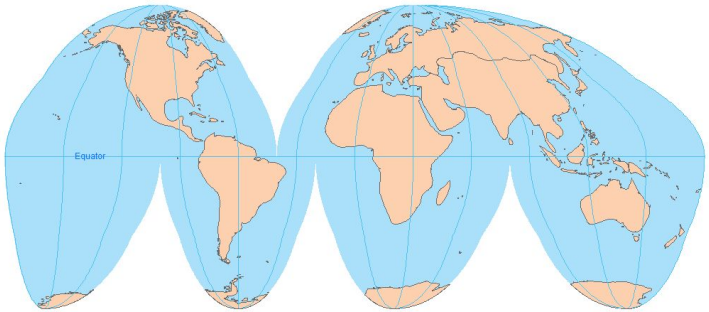






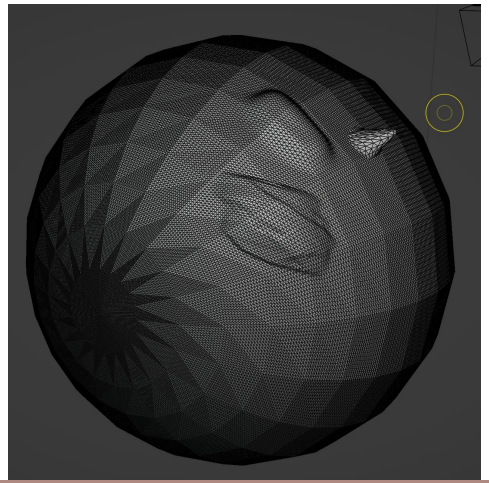
UV Coordinates: Assignment

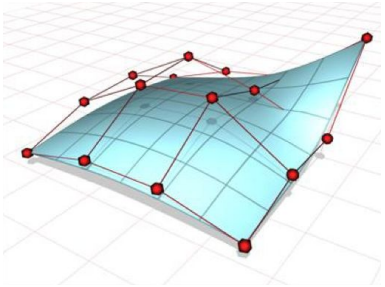
- Now, we have a list of vertex positions, a list of UV coordinates, and a list of indices into the vertex + UV list
- A single vertex might map to **multiple texture coordinates**
  - Can’t simply assign a UV coordinate to each vertex position
  - E.g. south pole

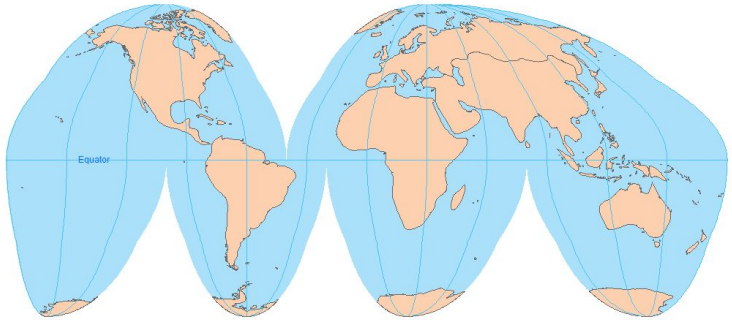


UV Coordinates: UV Unwrapping

- Create seams, then project/flatten/warp
  - Aka “unwrap” the object







UV Coordinates: UV Unwrapping

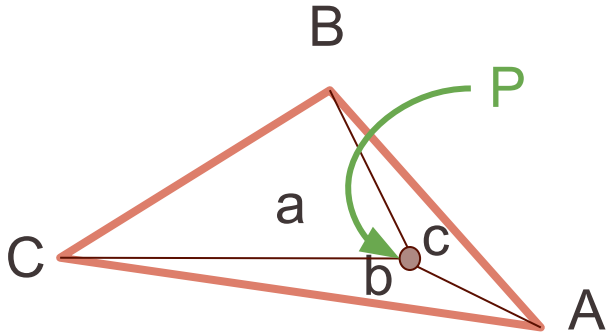
- How do we get 2D coordinates?
  - Per-vertex coordinates → surface?
- How do we **get** and **use** the lookup table?

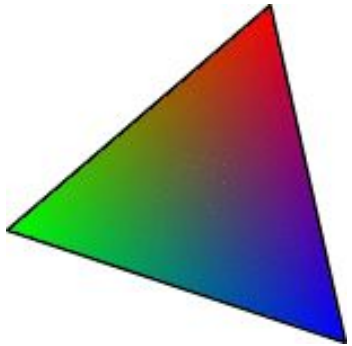
Lecture Outline

- Wrapping up ray tracing
  - Area lights + sampling, depth of field, color bleeding, global illumination, hybrid methods
- Textures overview
- Texture mapping
  - UV coordinates
- Texture lookup
  - Interpolation + sampling
- Normal mapping

Barycentric Interpolation

- Method to spatially interpolate between vertices to triangle interior
  - Recall lecture 3 (vertex and fragment shaders)



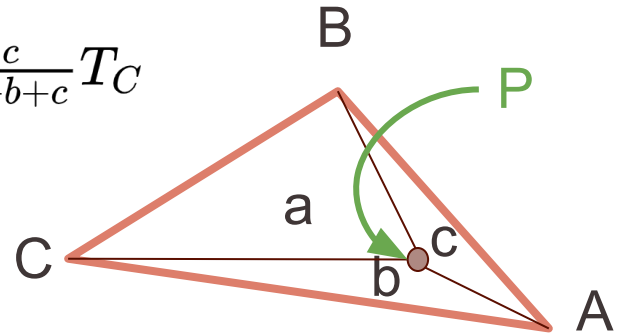


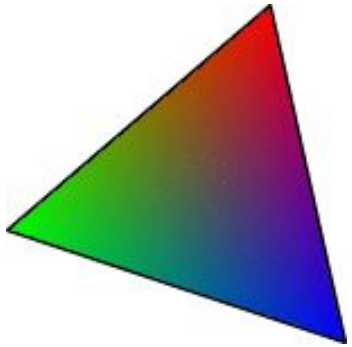
Barycentric Interpolation

- Method to spatially interpolate between vertices to triangle interior
  - Recall lecture 3 (vertex and fragment shaders)

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

The (u,v) coordinate for vertex A



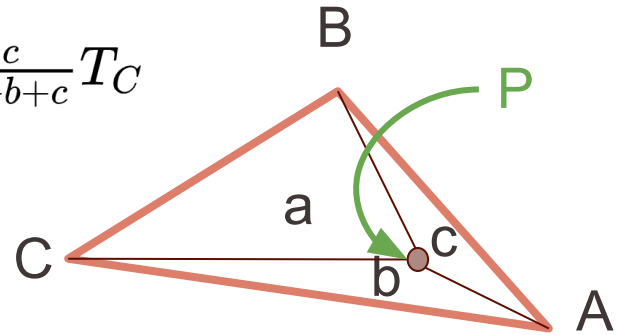


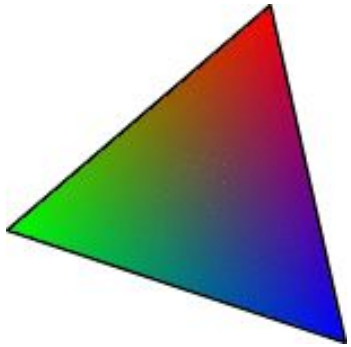
Barycentric Interpolation

- Method to spatially interpolate between vertices to triangle interior
  - Recall lecture 3 (vertex and fragment shaders)
- For raytracing: correct barycentric values in 3D space
- For rasterization: naive algorithm on perspective projection (lecture 4) actually gives incorrect values

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

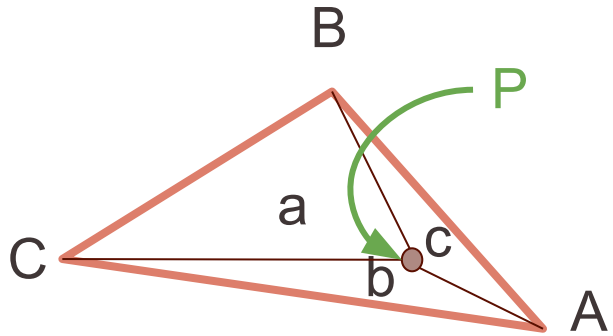
The (u,v) coordinate for vertex A





Screen Space Barycentric Interpolation

- Perspective correct interpolation
  - We need to address nonlinearity caused by the z component (depth) division



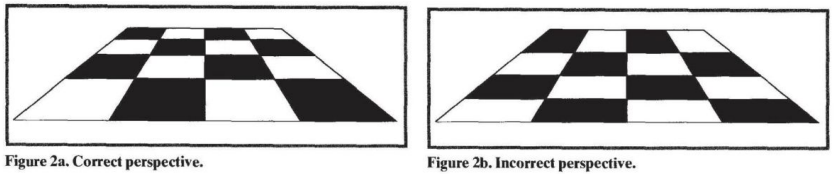


Figure 2a. Correct perspective.

Figure 2b. Incorrect perspective.

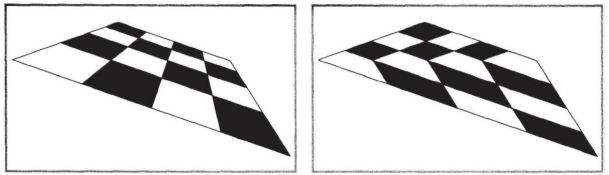


Figure 3a. Correct perspective.

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)!

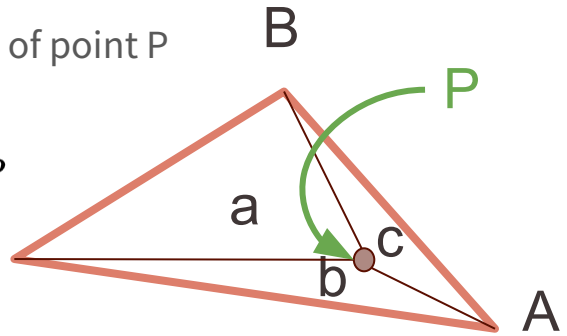
Screen Space Barycentric Interpolation

- Perspective correct interpolation
  - We need to address nonlinearity caused by the z component (depth) division

Depth (z-coord) of point P

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

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



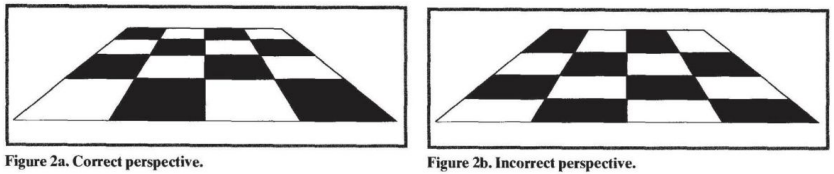


Figure 2a. Correct perspective.

Figure 2b. Incorrect perspective.

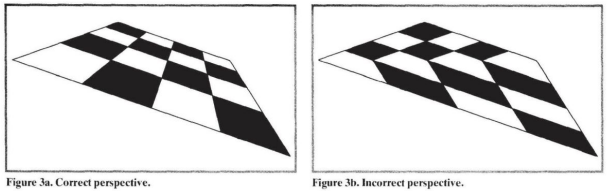


Figure 3a. Correct perspective.

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)!

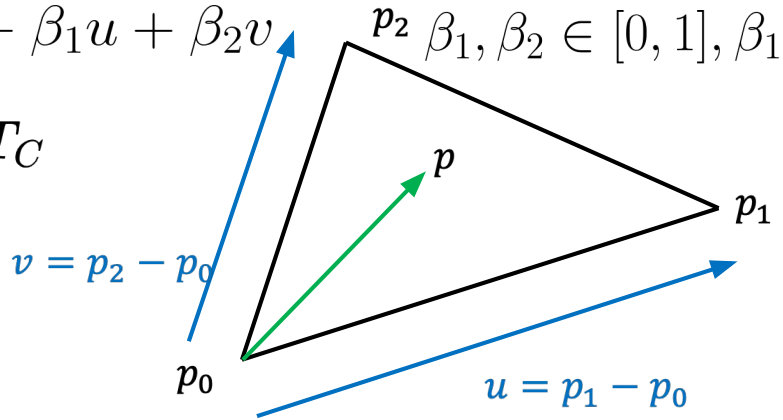
Barycentric Interpolation

- Method to spatially interpolate between vertices to triangle interior
  - Recall lecture 3 (vertex and fragment shaders)
- For raytracing: correct barycentric values in 3D space
- For rasterization: naive algorithm on perspective projection (lecture 4) actually gives incorrect values

$$p = p_o + \beta_1 u + \beta_2 v \qquad \beta_1, \beta_2 \in [0,1], \beta_1 + \beta_2 \leq 1$$

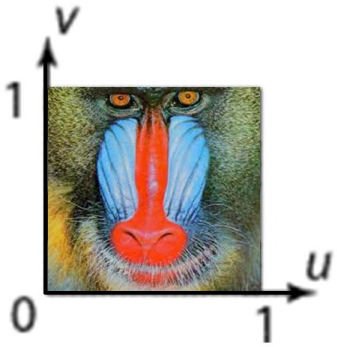
$$T_p = \frac{a}{a+b+c} T_A + \frac{b}{a+b+c} T_B + \frac{c}{a+b+c} T_C$$

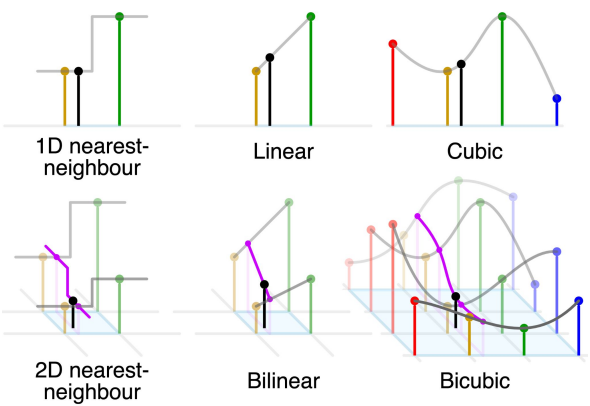
The (u,v) coordinate for vertex A

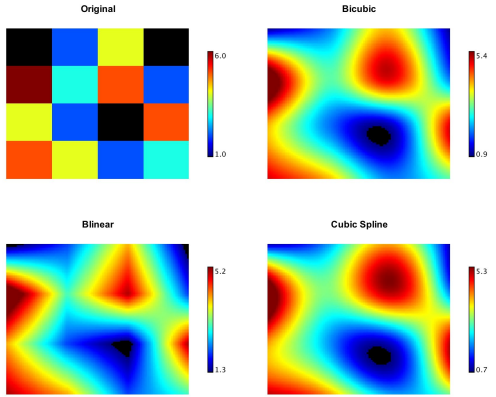


UV Coordinates to 2D Grid Lookup

- How to go from UV coordinates (floating point) to 2D pixel grid lookup
- 2D grid is **quantized**, so we need interpolation

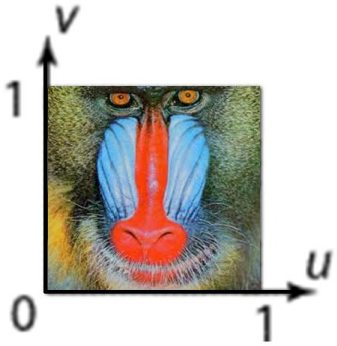


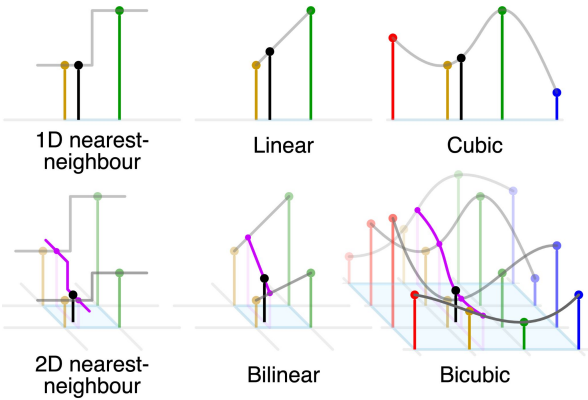


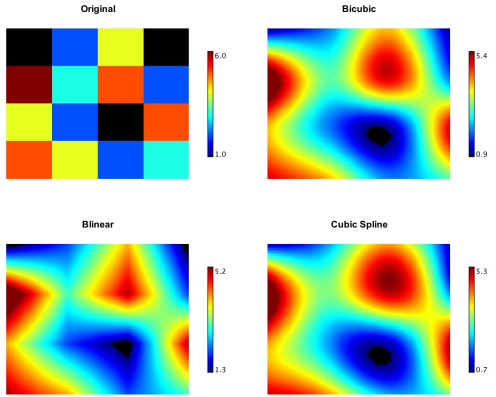


UV Coordinates to 2D Grid Lookup

- How to go from UV coordinates (floating point) to 2D pixel grid lookup
- 2D grid is **quantized**, so we need interpolation
- Example: 100x100 pixel RGB texture
  - UV coordinate of (0.253, 0.750) maps to (**25.3**, 75) in image space
  - Then we have to resolve the **25.3** (e.g. splines from lecture 2)







UV Coordinates to 2D Grid Lookup

- High resolution images may have aliasing
- Example: a 10000x10000 pixel RGB texture (10000x10000 **texels**)
  - UV coordinates of geometry rendered in two neighboring pixels will **map to texels far away**
  - E.g. pixel (100,50) and pixel (101,50) will map to very different spots on texture

UV Coordinates to 2D Grid Lookup

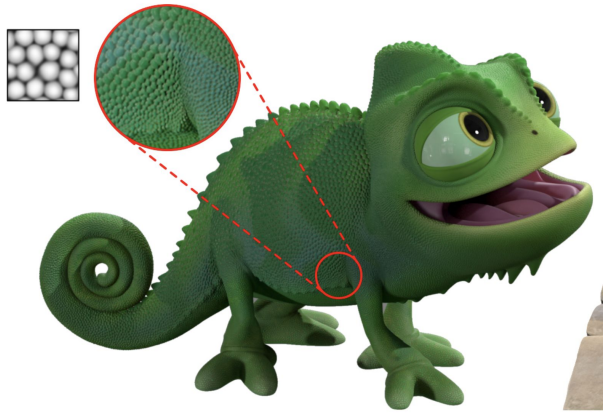
- High resolution images may have aliasing
- Example: a 10000x10000 pixel RGB texture (10000x10000 **texels**)
  - UV coordinates of geometry rendered in two neighboring pixels will **map to texels far away**
  - E.g. pixel (100,50) and pixel (101,50) will map to very different spots on texture
- Point sampling can’t fix this
  - These techniques rely on interpolating between a small number of nearby points (texel)

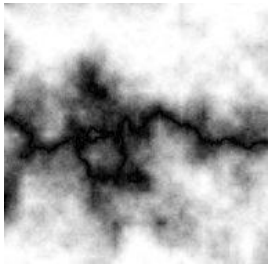
UV Coordinates to 2D Grid Lookup

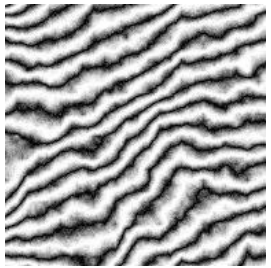
- Solution: store textures at multiple scales
  - Precompute the texture at multiple resolutions ahead of time
  - **Resolution-dependent lookup** (e.g. mipmaps)
  - If a pixel covers a huge chunk of the original texture, then sample from a coarser MIP
  - Optimal with 1:1 pixel:texture mapping (**texel**)

Texture Acquisition: Art, Capture, Procedural

- Many ways to obtain textures
- Artist crafted textures
  - Aided with specialized software
- Real-world captured data
- **Procedural textures**
  - Crafted for generic use

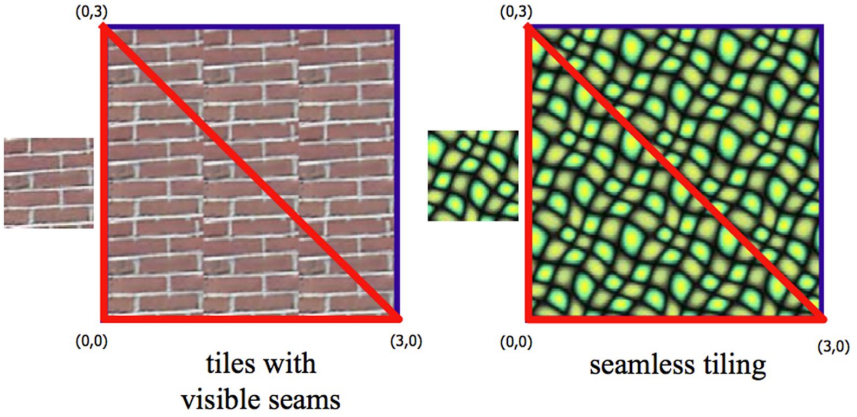






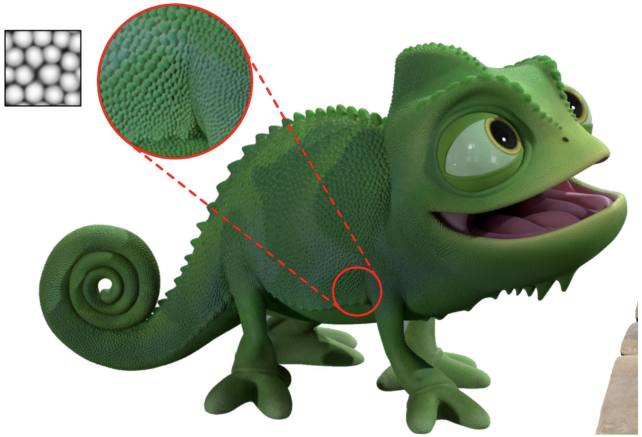
Generalized Seamless Textures

- Can add interesting patterns / repeated details via seamless patterns
- UV coordinates don't have to be 0-1
  - We can also wrap around for high resolution detail without needing high resolution textures
  - Example: UV coordinate (1.2,2.5) maps to (0.2,0.5) on the lookup grid



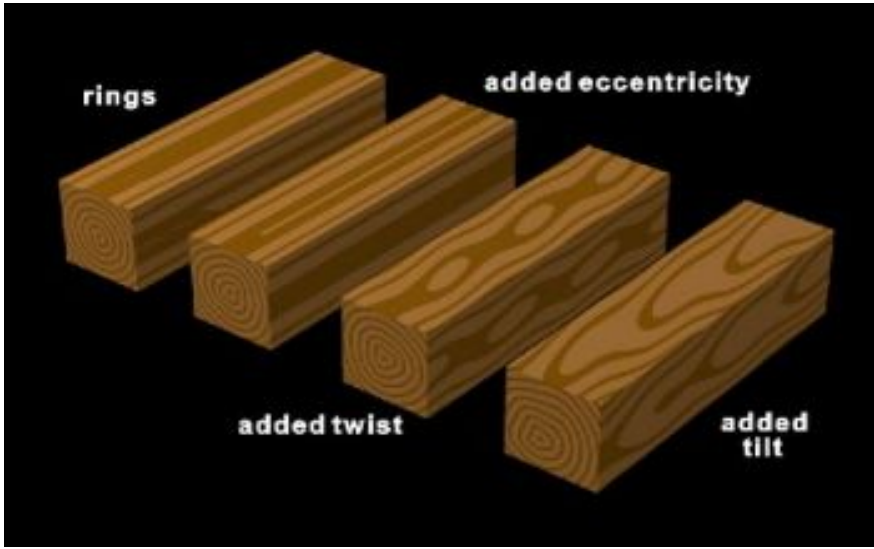
Generalized Seamless Textures

- Can add tiny repeating details via **seamless** patterns
- These can be artist generated OR procedurally generated
- You can try out these textures on the homework:
  - https://renderman.pixar.com/pixar-one-thirty



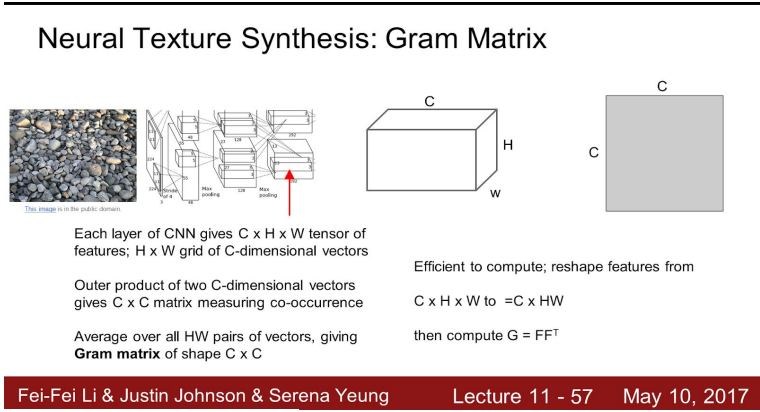
Procedural Textures

- Can be explicitly modeled (e.g. 3D wood textures)
  - Note: we use 2D textures a lot because we deal with surfaces, but 3D texture lookups are valid, too
- Can be modeled with random noise (Perlin/Simplex noise)



Procedural Textures

- Can be created with ML/AI models!



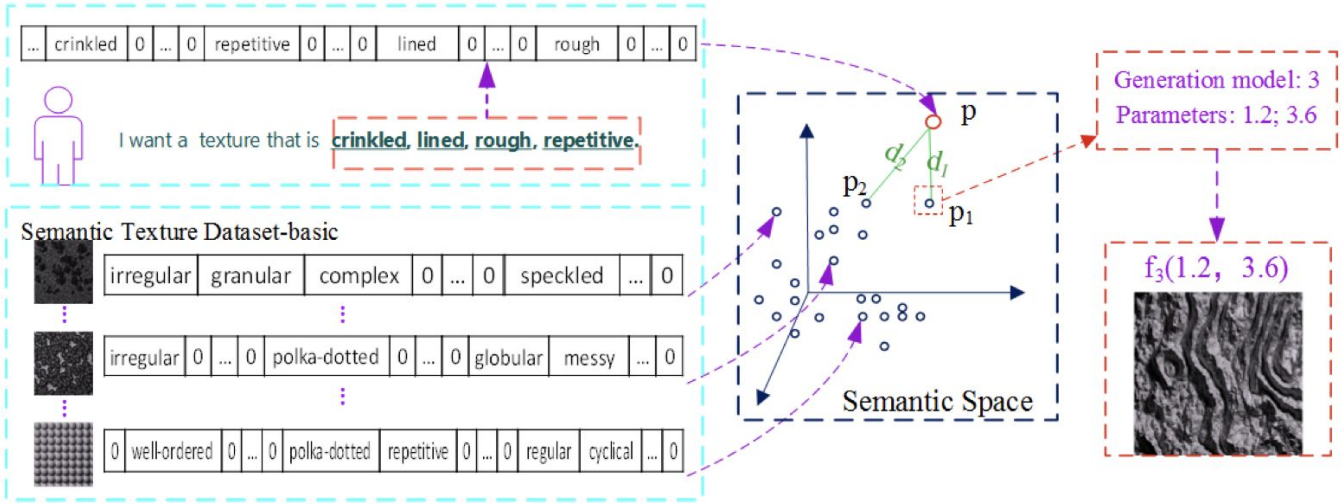


Fig. 4: The workflow of our texture generation process.

https://arxiv.org/pdf/1704.04141

Lecture Outline

- Wrapping up ray tracing
  - Area lights + sampling, depth of field, color bleeding, global illumination, hybrid methods
- Textures overview
- Texture mapping
  - UV coordinates
- Texture lookup
  - Interpolation + sampling
- Normal mapping

Normal Maps

- Recall: think about textures as a lookup table for BxDF (not as “giftwrap”)
- One lookup table per BxDF parameter that needs to be spatially varying
- Example: diffuse map, specular map, normal map

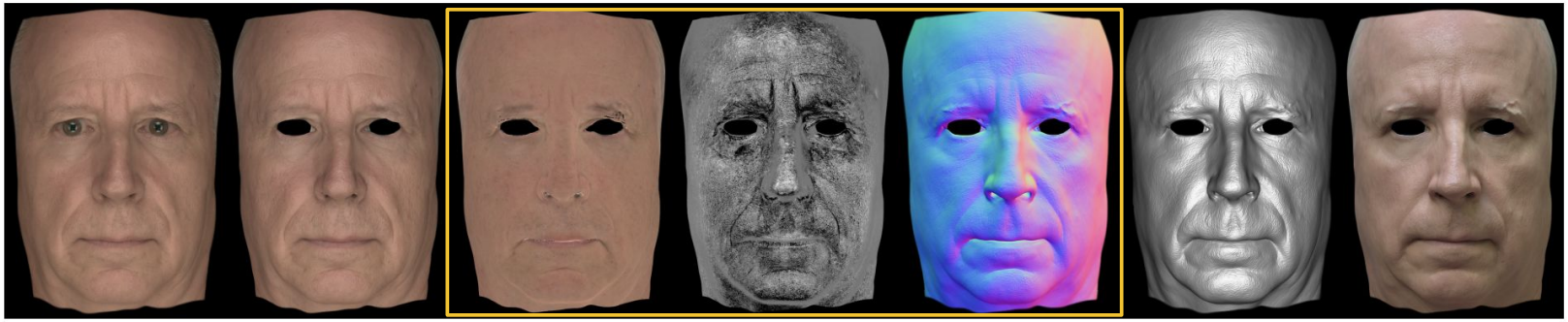
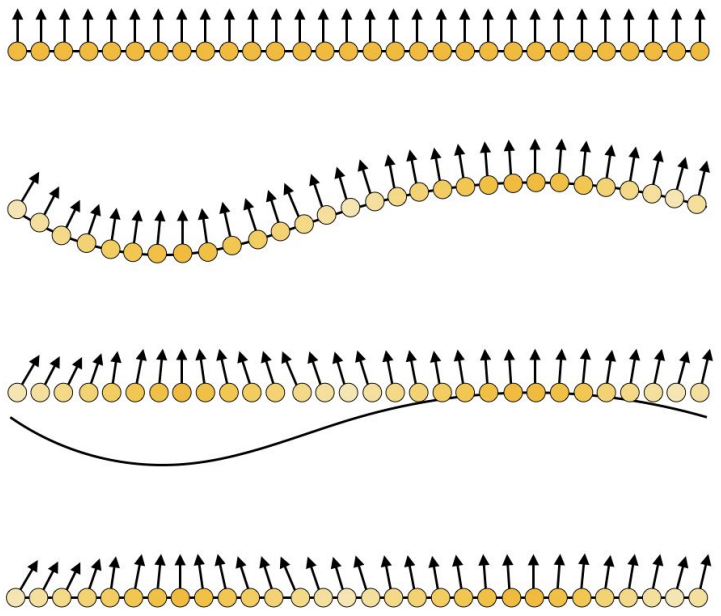


Fig. 1. We present a method to capture complete facial geometry and appearance from a *single* exposure. From left to right: one input image, our matching render, diffuse albedo, specular intensity, normals, high resolution geometry, and a realistic re-render under a different environment map.

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

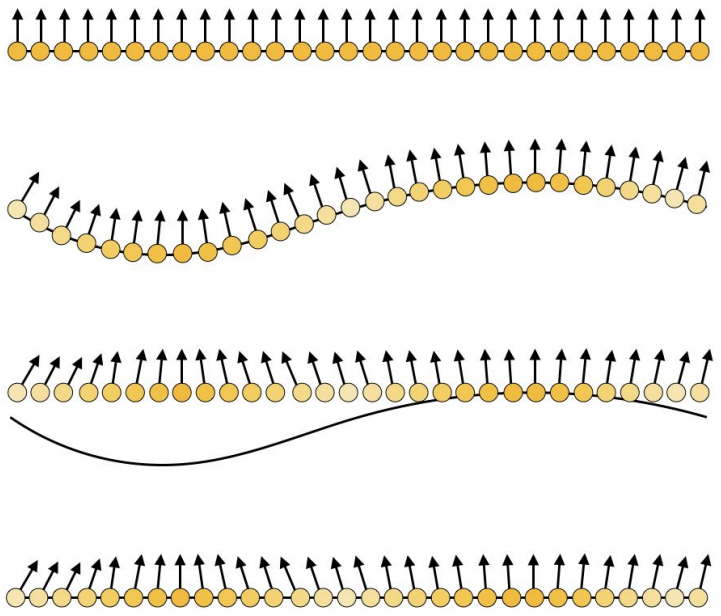
Normal Maps

- Goal: change the normals on some parts of the textures
  - normal will hit the light differently (sometimes perpendicular, sometimes not)



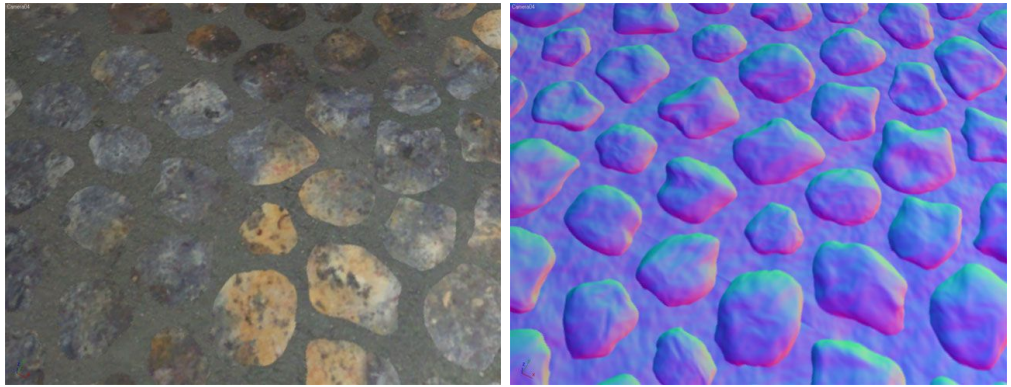
Normal Maps

- Goal: change the normals on some parts of the textures
  - normal will hit the light differently (sometimes perpendicular, sometimes not)
- Now all we need is **one triangle**, but it looks different because of the normal map of the texture



Normal Maps

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



diffuse map

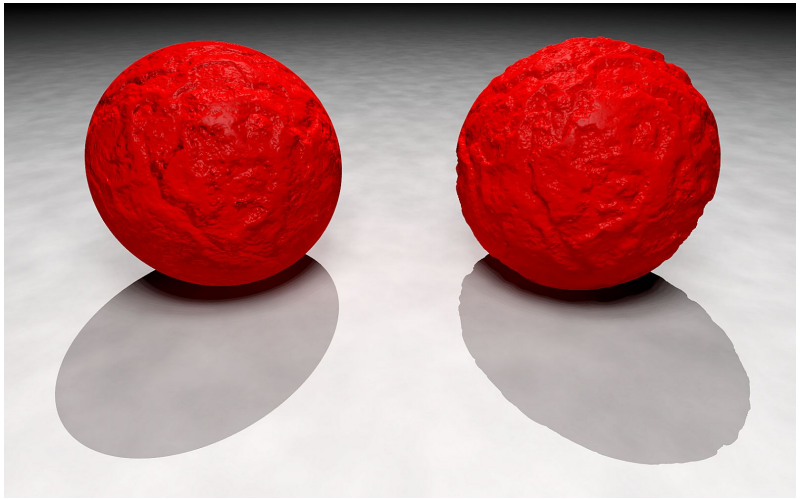
normal map

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

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

Normal Maps

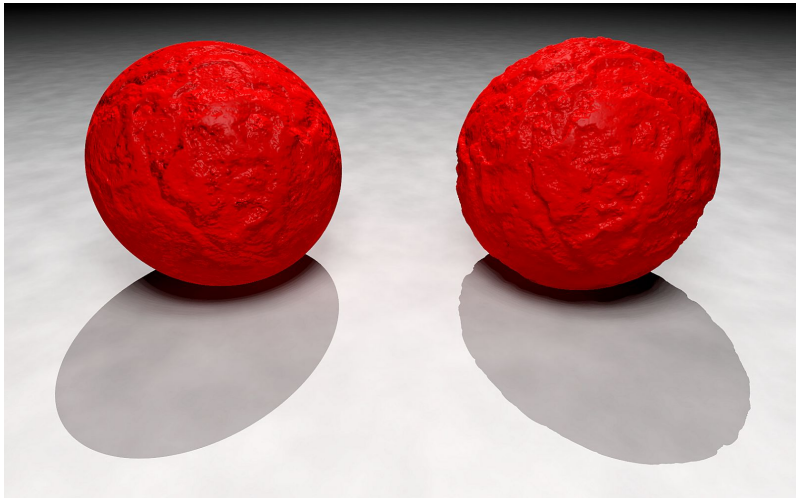
- Other techniques:
  - **Bump mapping**: 1D height map (greyscale) instead of 3D normals (rgb)



Bump mapping on the left,
displacement mapping on the right

Normal Maps

- Other techniques:
  - **Bump mapping**: 1D height map (greyscale) instead of 3D normals (rgb)
  - **Displacement mapping**: on-the-fly subdivision (lecture 2) and modification of geometry



Bump mapping on the left,
displacement mapping on the right

So much work!

- Requires texturing tools
  - Computational geometry algorithms
  - Specialized user interfaces
  - Artist expertise

So much work!

- Requires texturing tools
  - Computational geometry algorithms
  - Specialized user interfaces
  - Artist expertise
- Specialized software
  - All-in-1 (Blender) vs. Specialized
  - You probably have a sense by now of how hard it is to build all-in-1 graphics tools
  - Just “rendering” itself requires a lot of infrastructure (lectures 5-7)
- Blender/Maya is more than enough most of the time!

Congrats!!

- First half: crash course in many topics. Second half: fun/advanced topics, final project workshops, and guest speakers.

	Tuesday Lecture	Thursday Lecture	Homework due (Thursday)
Week 1	Introduction	Geometry & Transformations	*Installation & Setup* *
Week 2	Rasterization & Shading	Color, Images & Cameras	Geometry & Transformations
Week 3	Light & Optics	Raytracing I	Shading & Cameras
Week 4	Raytracing II	Sampling & Texturing	Raytracing
Week 5	Final Project Expectations	Simulation & Animation I	Lighting & Texturing
Week 6	Simulation & Animation II	Guest Lecture	Simulation & Advanced Rendering
Week 7	Advanced Topics	Final Project Workshop	
Week 8	Art of Images	Next Steps in Graphics	Project Submission

Additional Resources + Exploring!

- Turner Whitted
- NVIDIA: What is Path Tracing?
- Snow White: Mirror Man
- Linear Methods for Image Interpolation
- A Procedural Texture Generation Framework Based on Semantic Descriptions
- Skin Appearance Capture

- Fundamentals of Computer Graphics, Steve Marschner and Peter Shirley
  - Preview, Book

Sampling and Textures AND TRIVIA!

Lecture 1: Introduction

- Q: Which animated movie was the first to render curly hair?

Lecture 1: Introduction

- Q: Which animated movie was the first to render curly hair?
- A: Brave, Pixar 2012

Lecture 2: Geometry and Transformations

- Q: Why is this bunny relevant in graphics? Bonus points if you know approximately how many triangles comprise it.

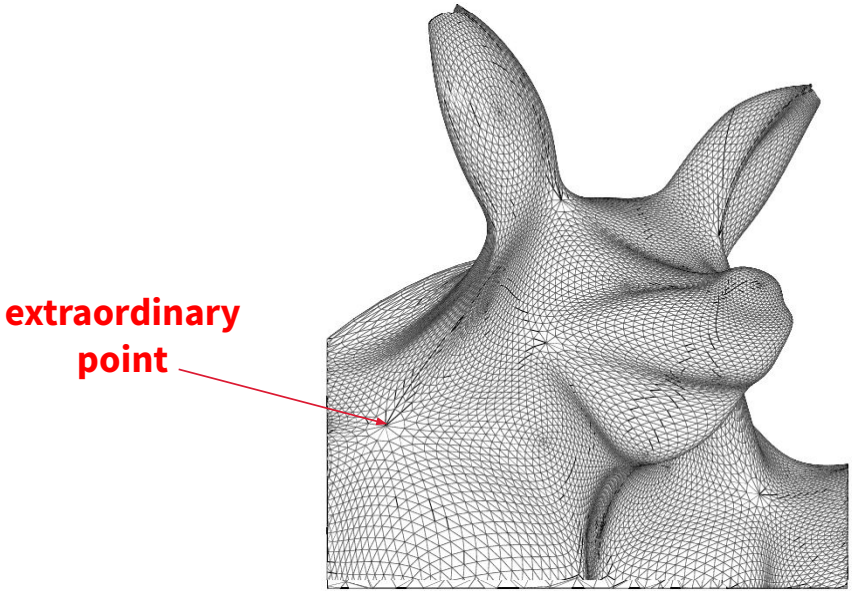
Lecture 2: Geometry and Transformations

- Q: Why is this bunny relevant in graphics? Bonus points if you know approximately how many triangles comprise it.
- A: The Stanford Bunny – a famous computer graphics 3D test model

Stanford Bunny
69,451 triangles

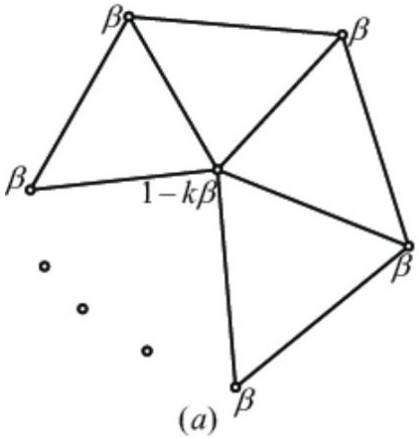
Lecture 2: Geometry and Transformations

- Q: How do we deal with extraordinary points in loop subdivision?

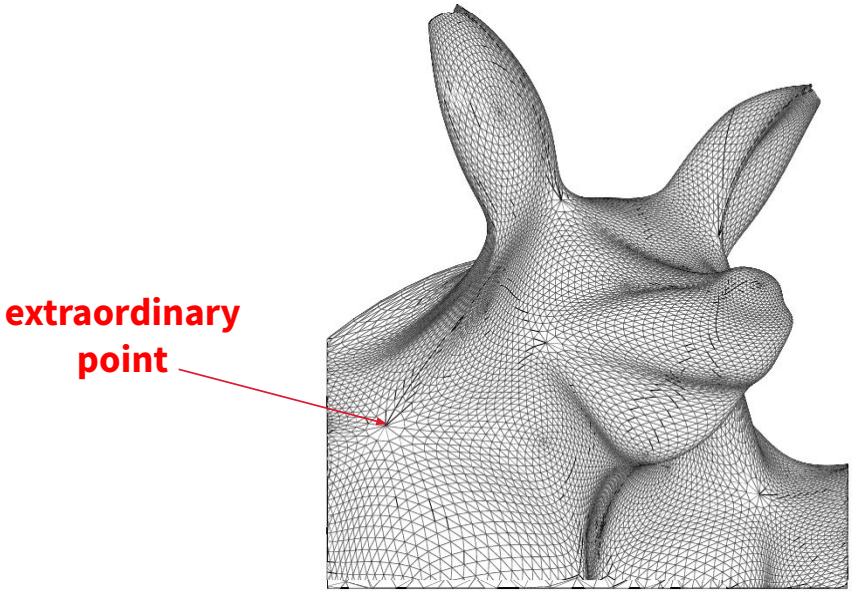


Lecture 2: Geometry and Transformations

- Q: How do we deal with extraordinary points in loop subdivision?
- A: Weight the vertex by the number of surrounding vertices



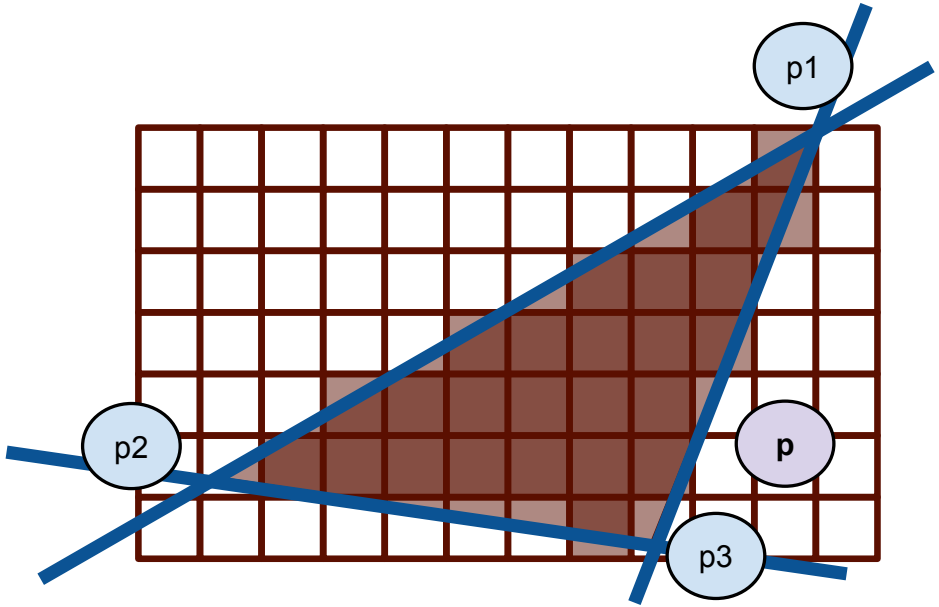
$$\beta = \frac{3}{16}, n = 3 \qquad \beta = \frac{3}{8n}, n \neq 3$$



Lecture 3: Rasterization and Shading

- Q: Which of these sign functions will be negative? Positive? Hint: use the right hand rule.

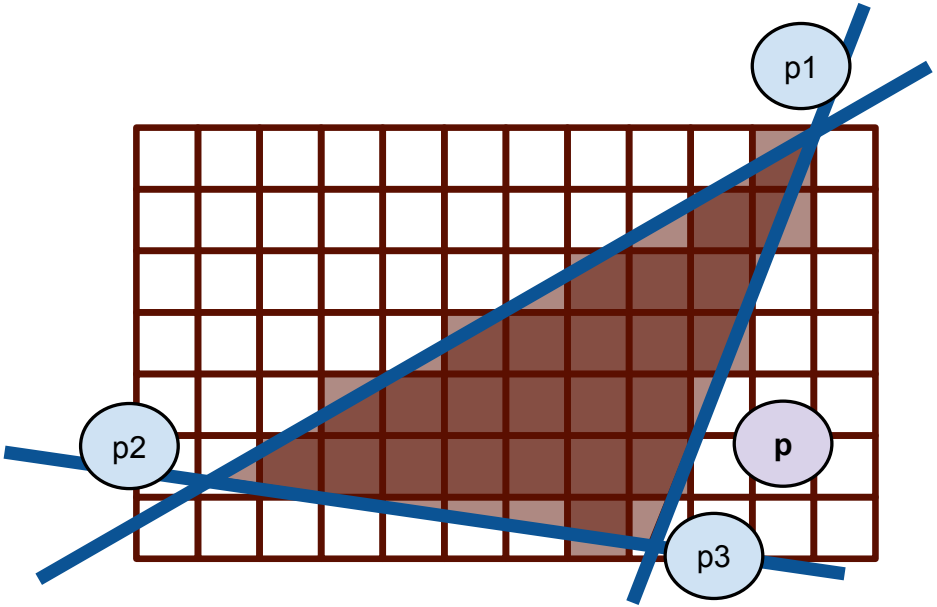
$$S1(p) = (p - p_1) \times (p_2 - p_1)$$
$$S2(p) = (p - p_2) \times (p_3 - p_2)$$
$$S3(p) = (p - p_3) \times (p_1 - p_3)$$



Lecture 3: Rasterization and Shading

- Q: Which of these sign functions will be negative? Positive? Hint: use the right hand rule.
- A:
  - S1(p) < 0
  - S2(p) < 0
  - S3(p) > 0

$$S1(p) = (p - p_1) \times (p_2 - p_1)$$
$$S2(p) = (p - p_2) \times (p_3 - p_2)$$
$$S3(p) = (p - p_3) \times (p_1 - p_3)$$



Lecture 3: Rasterization and Shading

- Q: Briefly describe the 3 components of the Phong reflection model. Which one is the most complex to calculate?

Lecture 3: Rasterization and Shading

- Q: Briefly describe the 3 components of the Phong reflection model. Which one is the most complex to calculate?

- A:
  - **Ambient:** “Base color”.
  - **Diffuse:** “Rough material”. Given the same light source, objects look brighter when hit “perpendicularly”.
  - **Specular:** “Shiny material”. Bright highlights when light reflects into our eyes.

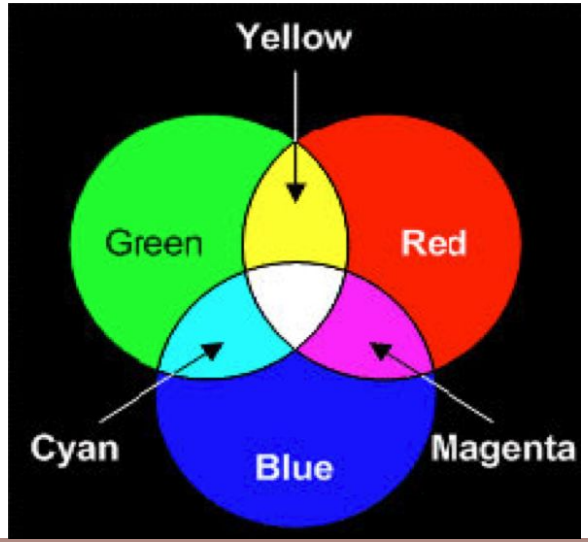
C_i = ambient + diffuse + specular

Lecture 4: Color, Images, Cameras

- Q: Why do printers use yellow, cyan, and magenta?

Lecture 4: Color, Images, Cameras

- Q: Why do printers use yellow, cyan, and magenta?
- A: We use subtractive color principles for ink on paper. We start with white, then each ink layer removes certain wavelengths by absorbing them (e.g. a cyan material absorbs red light).



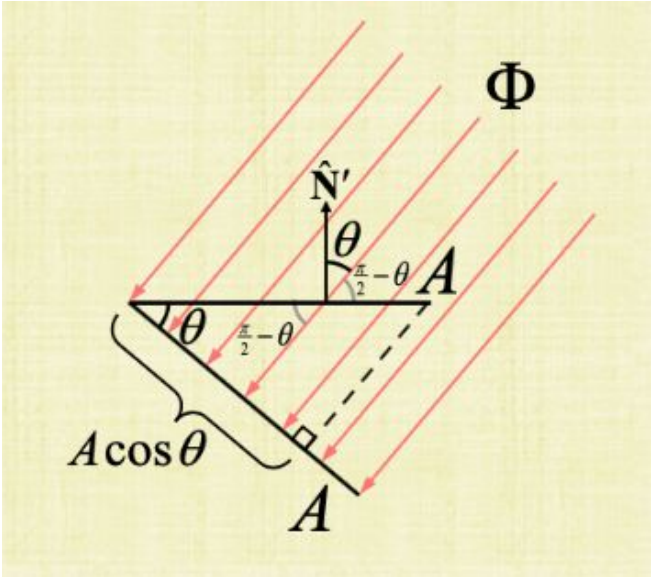
Lecture 5: Light and Optics

- Q: How does irradiance change as you tilt a surface?

Lecture 5: Light and Optics

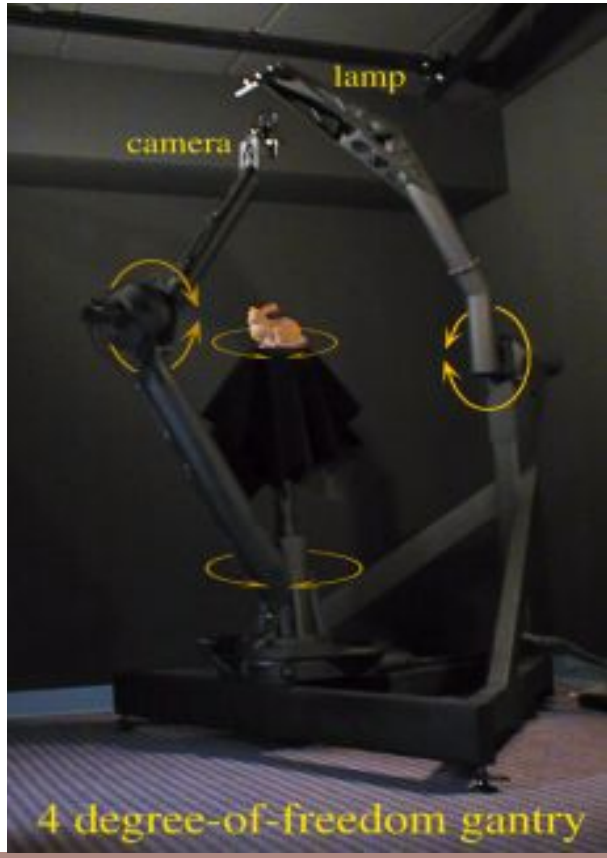
- Q: How does irradiance change as you tilt a surface?
- A: Irradiance decreases as you tilt the surface.

$$E = \frac{dP}{dA} \rightarrow E_{tilted} = \frac{\frac{A\cos\theta}{A}P}{A} = E\cos\theta$$



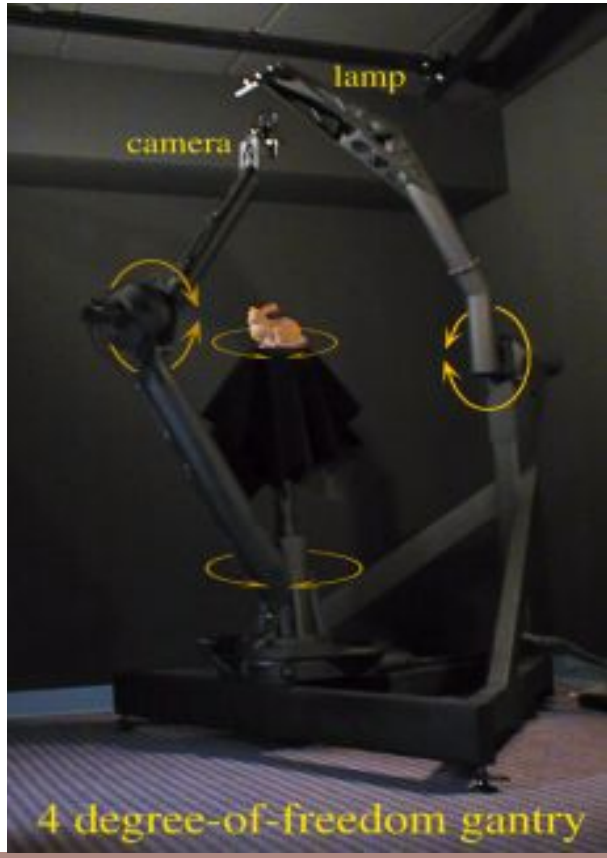
Lecture 5: Light and Optics

- Q: What does this machine do? Bonus points for its name.



Lecture 5: Light and Optics

- Q: What does this machine do? Bonus points for its name.
- A: A gonioreflectometer. Captures an object's BRDF information by shining light on it and observing its reflective properties.



Lecture 5: Light and Optics

- Q: Why is the integral version of the lighting equation preferred over the discrete summation version?

$$L_o(\omega_o) = \sum_{i \in in} L_o(\omega_i, \omega_o)$$

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

Lecture 5: Light and Optics

- Q: Why is the integral version of the lighting equation preferred over the discrete summation version?
- A: The integral version allows us to “sum” over solid angles. This allows us avoid identifying every possible source of light and finding its discrete contribution.

$$L_o(\omega_o) = \sum_{i \in in} L_o(\omega_i, \omega_o)$$

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

Lecture 6: Ray Tracing Part 1

- Q: In this second (preferred) method of solving ray-object intersection, what do B1, B2, and t represent?

$$(u, v, A - P) \begin{pmatrix} \beta_1 \\ \beta_2 \\ t \end{pmatrix} = A - p_o$$

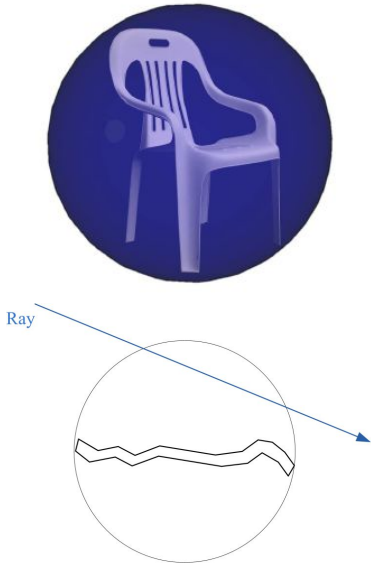
Lecture 6: Ray Tracing Part 1

- Q: In this second (preferred) method of solving ray-object intersection, what do B1, B2, and t represent?
- A:
  - B1 = scaling of the u edge vector of the triangle
  - B2 = scaling of the v edge vector of the triangle
  - t = length of the ray from the camera aperture to the intersection point

$$(u, v, A - P) \begin{pmatrix} \beta_1 \\ \beta_2 \\ t \end{pmatrix} = A - p_o$$

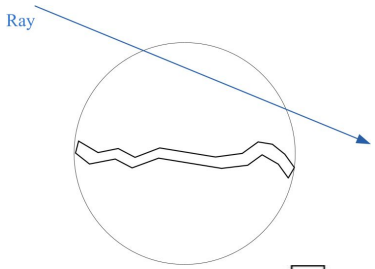
Lecture 6: Ray Tracing Part 1

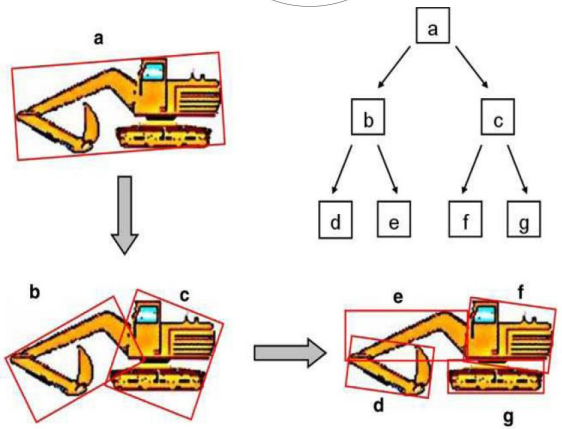
- Q: Why do we use bounding volumes during ray tracing? Why is bounding volume hierarchy an even better solution?



Lecture 6: Ray Tracing Part 1

- Q: Why do we use bounding volumes during ray tracing? Why is bounding volume hierarchy an even better solution?
- A: Previously, we had to loop over all the triangles in our scene to test for intersection points. Bounding volumes allow us to surround objects in simpler shapes that we can check for intersection with first. BVH accelerates this process by using trees to represent the possible intersection volumes.



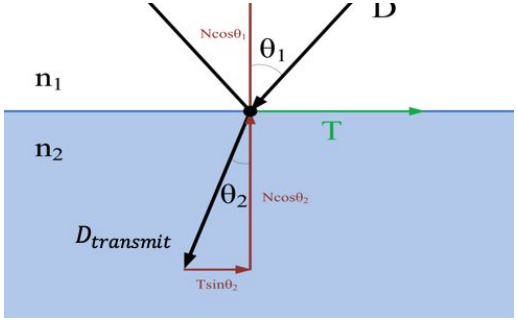


Lecture 7: Ray Tracing Part 2

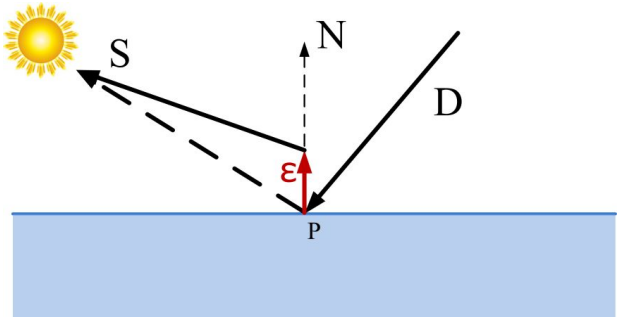
- Q: Which types of rays need corrections for spurious self-occlusion? What does this look like for each of these rays?

Lecture 7: Ray Tracing Part 2

- Q: Which types of rays need corrections for spurious self-occlusion? What does this look like for each of these rays?
- A: (1) Shadow rays and adding in normal direction, (2) reflection rays and adding in normal direction, (3) transmission rays and subtracting in normal direction



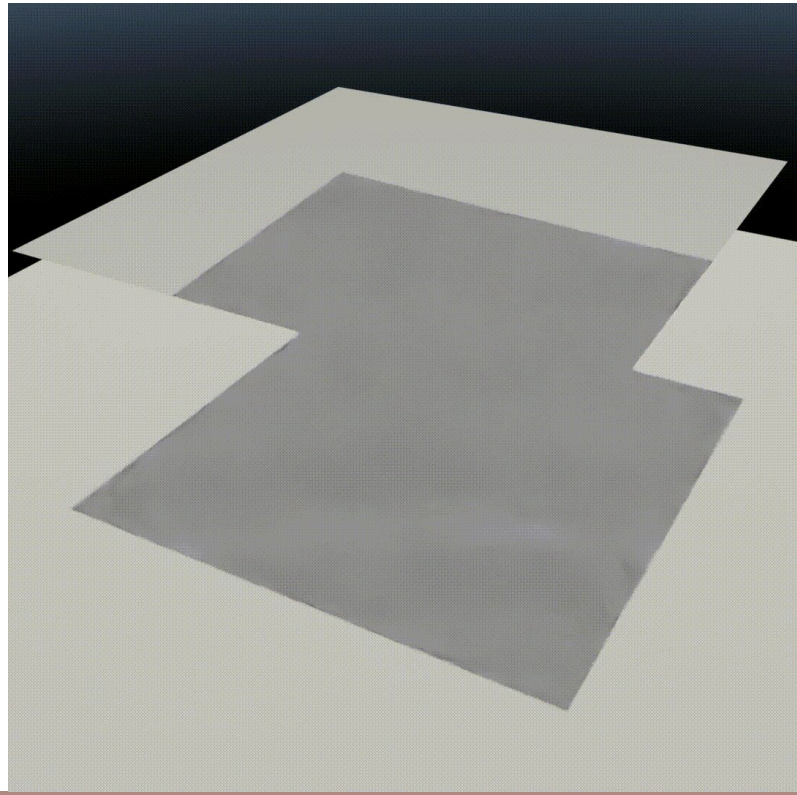
$$R_{transmit}(t) = \left(P - \epsilon\hat{N}\right) + \hat{D}_{transmit}\, t$$



$$R_{reflect}(t) = \left(P + \epsilon\hat{N}\right) + \hat{D}_{reflect}\, t$$

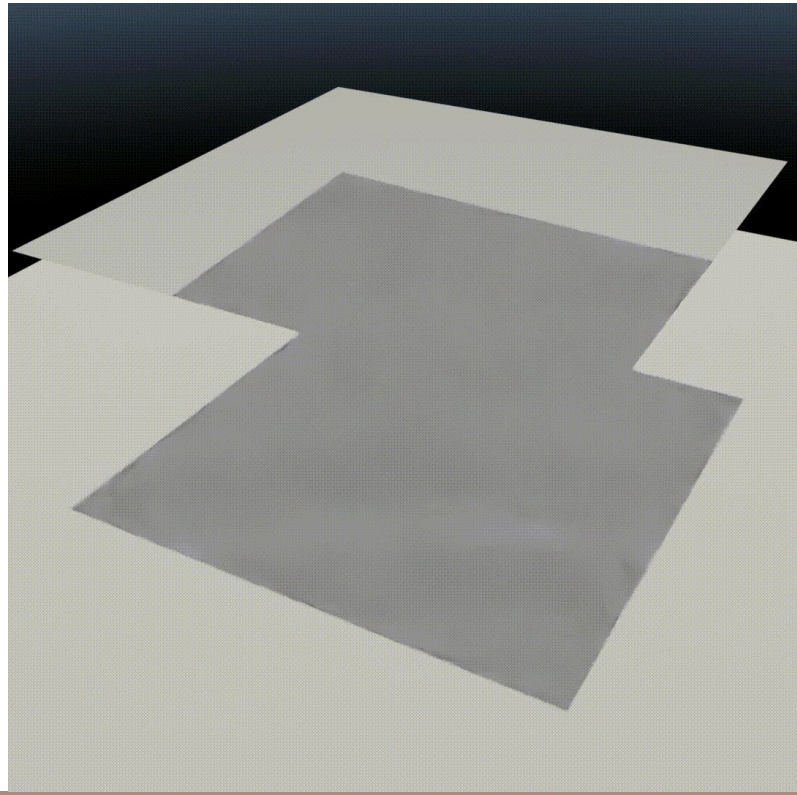
Lecture 7: Ray Tracing Part 2

- Q: How do we create the effect shown here?



Lecture 7: Ray Tracing Part 2

- Q: How do we create the effect shown here?
- A: Leverage caustics. This makes light leaving a material concentrate at certain points depending on the curvature of the material

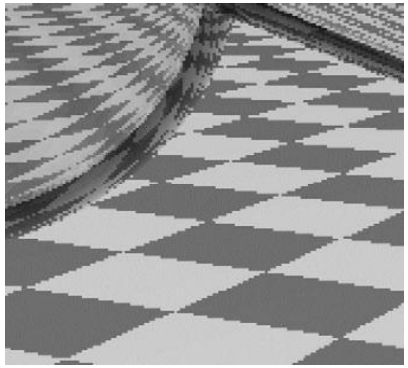


Lecture 8: Sampling and Textures

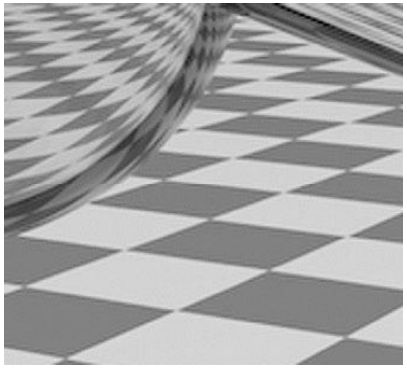
- Q: How does sampling camera rays prevent anti-aliasing?

Lecture 8: Sampling and Textures

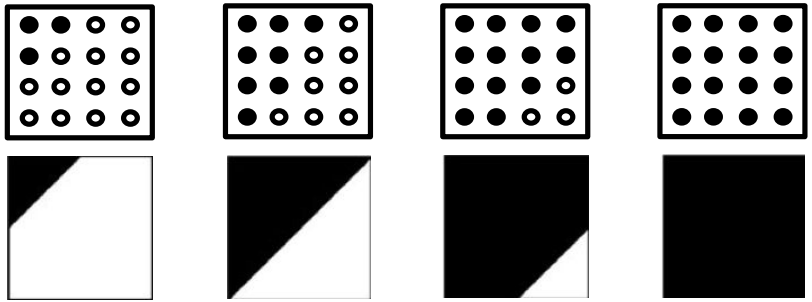
- Q: How does sampling camera rays prevent anti-aliasing?
- A: This process makes the pixel finer grained. We can then sample camera rays through random points in the pixel and average the results.



aliased



anti-aliased



Congrats!!

- First half: crash course in many topics. Second half: fun/advanced topics, final project workshops, and guest speakers.

	Tuesday Lecture	Thursday Lecture	Homework due (Thursday)
Week 1	Introduction	Geometry & Transformations	*Installation & Setup* *
Week 2	Rasterization & Shading	Color, Images & Cameras	Geometry & Transformations
Week 3	Light & Optics	Raytracing I	Shading & Cameras
Week 4	Raytracing II	Sampling & Texturing	Raytracing
Week 5	Final Project Expectations	Simulation & Animation I	Lighting & Texturing
Week 6	Simulation & Animation II	Guest Lecture	Simulation & Advanced Rendering
Week 7	Advanced Topics	Final Project Workshop	
Week 8	Art of Images	Next Steps in Graphics	Project Submission

Attendance