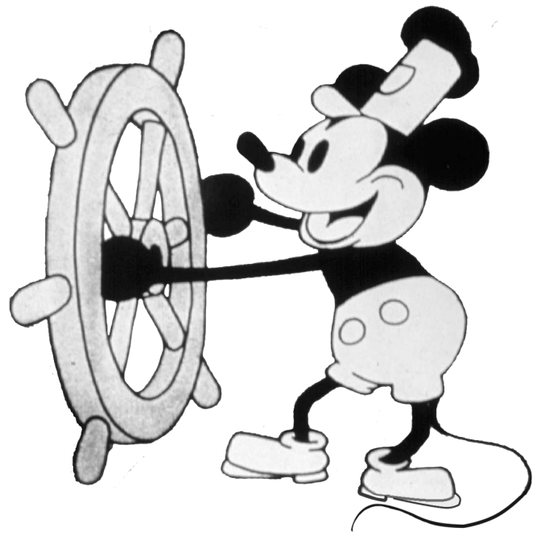


CSCI 461: Computer Graphics

Middlebury College, Fall 2025

Lecture 07: Texturing

Reminders about deadlines and use of AI.

- If no submission by initial deadline, no feedback at any point.
- Deadline for regrade requests is 2 business days before the final deadline.
- Submissions with pending grades will be regraded after the final deadline (no need to request a regrade).
- Final deadlines are **final** (unless there are extenuating circumstances).

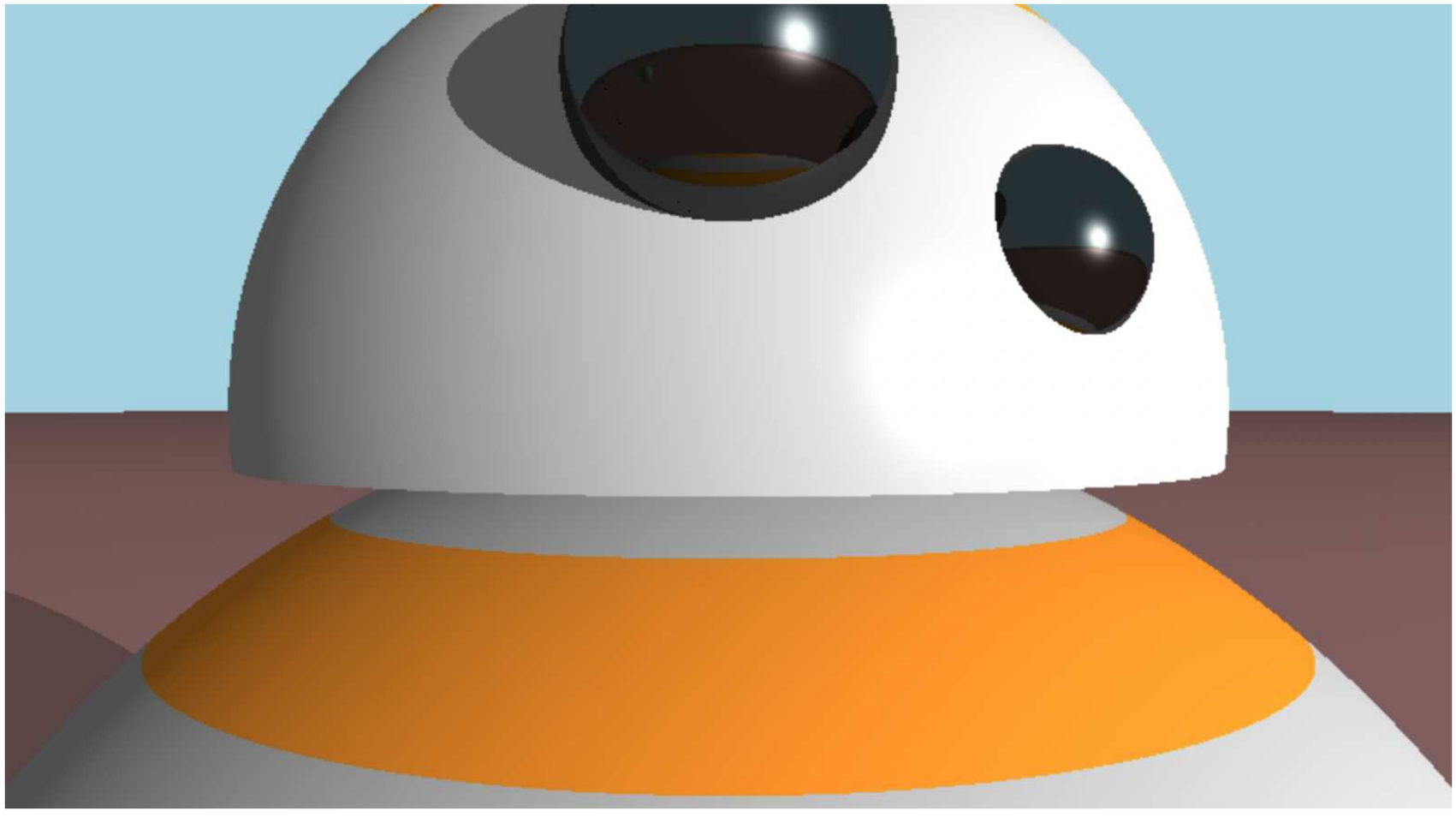
No feedback for submissions suspected of AI use.

- Initial feedback will be empty, and then I'll grade normally after the final deadline.
- If you think this is a mistake, then come talk to me about your submission.
- Final submissions using techniques we haven't covered will receive a final grade for that lab of "N" (Not Assessable).

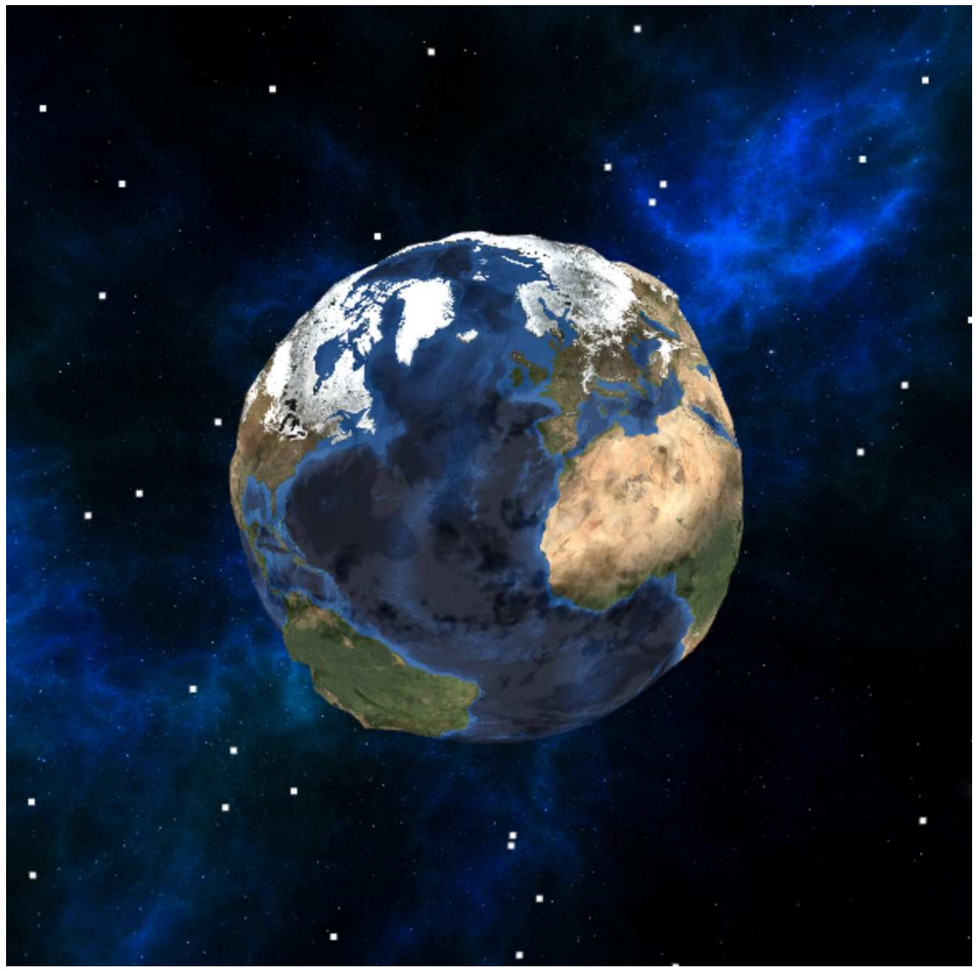
Final deadline for Lab 3 is tonight (10/21) at 11:59pm.

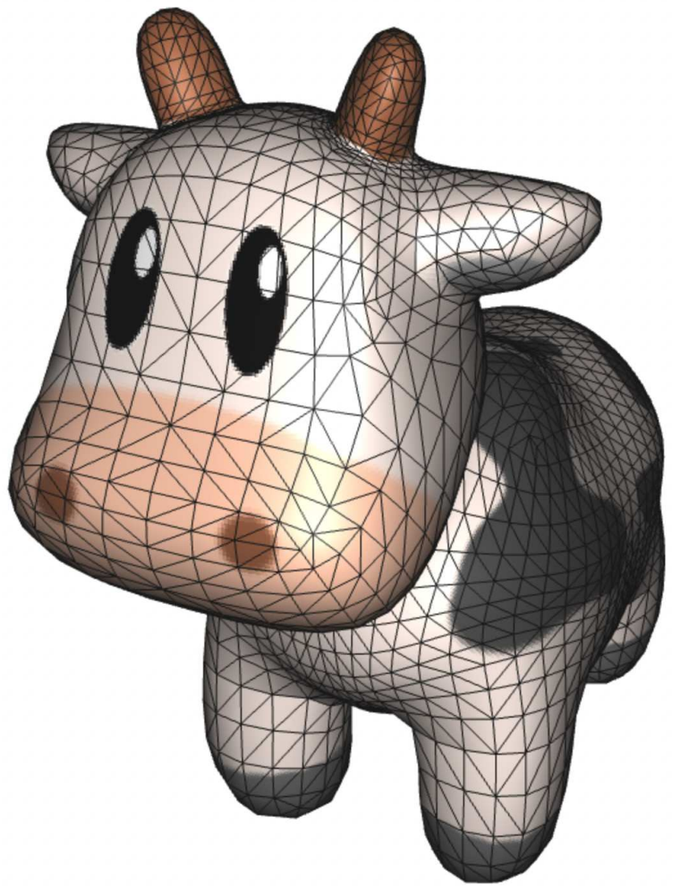
Recap on how we have been painting our models.

$$I = c_a k_m + c_l k_m \max\left(0, \vec{n} \cdot \vec{l}\right) + c_l k_s \max(0, \vec{v} \cdot \vec{r})^p$$



But what if we want to do this?

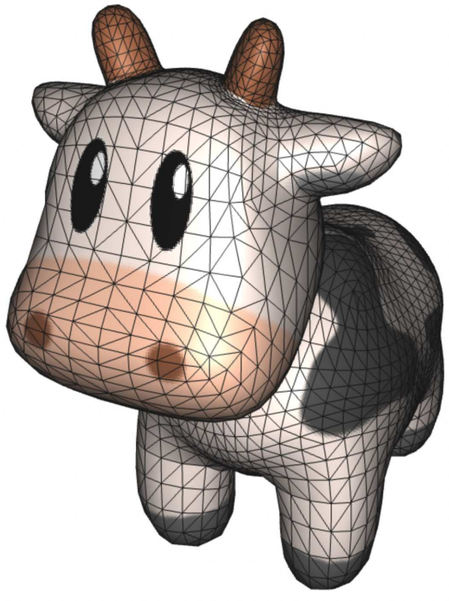


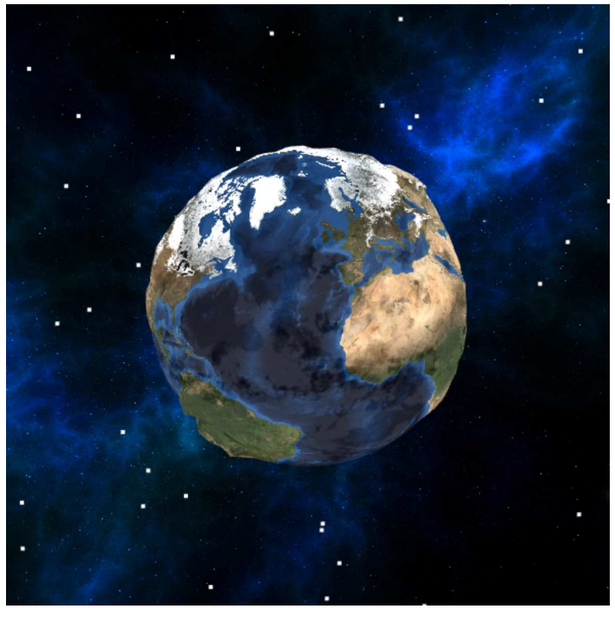


Two things we want: (1) color detail, (2) geometric detail.

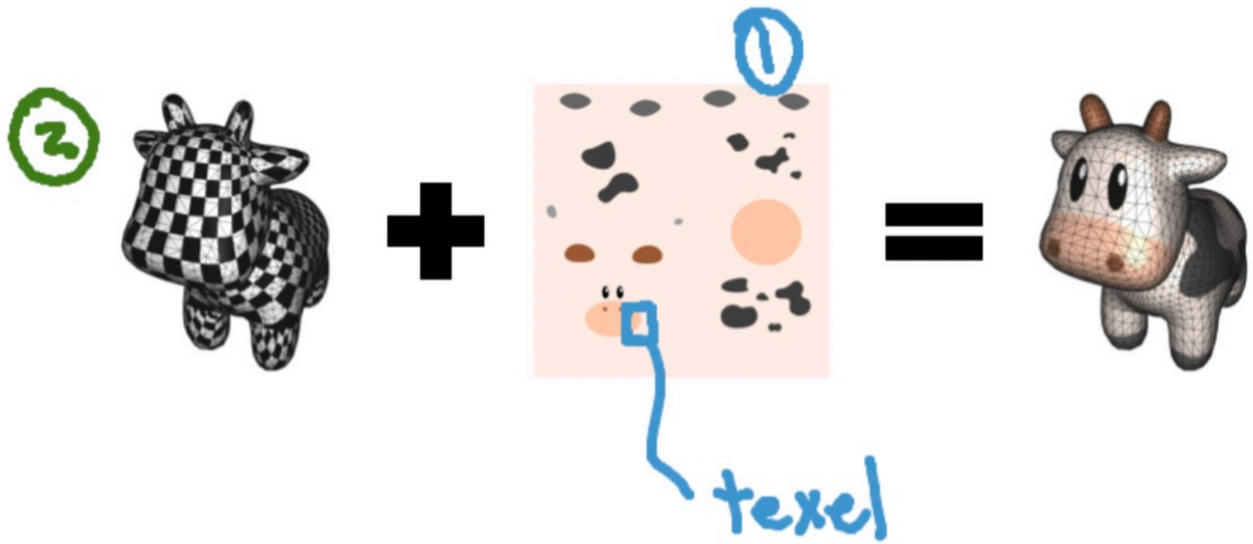
By the end of today's lecture, you will be able to:

- analytically calculate **texture coordinates** on a sphere,
- **sample texels** in a texture image to color a surface,
- add mouse **controls** to your renderings, 
- ray trace scenes which have model transformations.



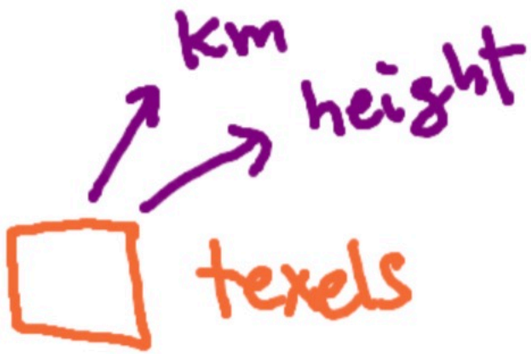


The main idea of texturing: sample an image to determine surface properties.

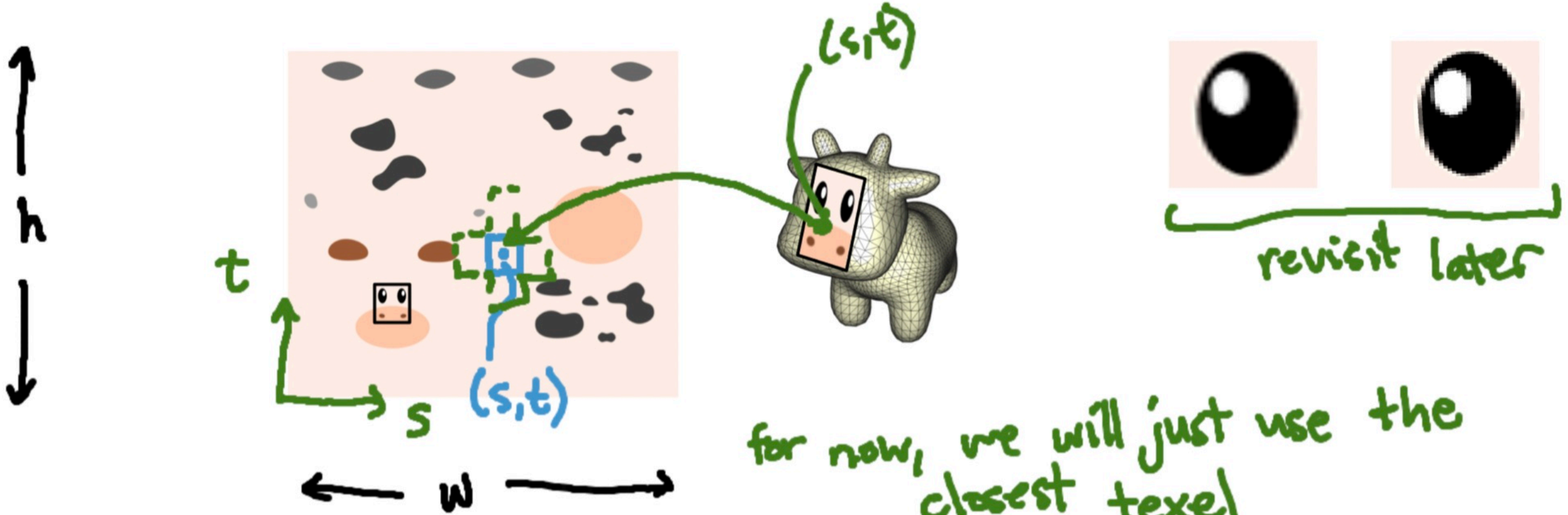


three ingredients:

1. image to paste
2. texture coordinates
3. a way to look up



Ingredients #1 and #3: which *texel* to sample?



for now, we will just use the closest texel

spot.png

```
1 <!-- in your HTML document -->
2 <img id="earth-color" src="earth.jpg" hidden />
3
4 <!-- it's also usually a good idea to wrap your rendering script in
5      "window.onload" to make sure the images have loaded -->
```

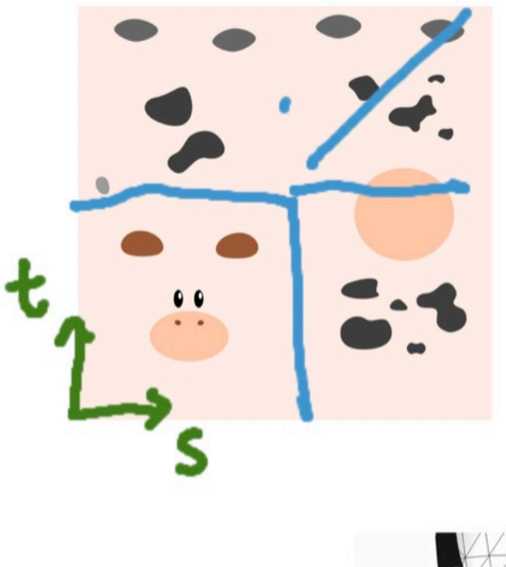
Ingredient #2: we need *texture coordinates*.

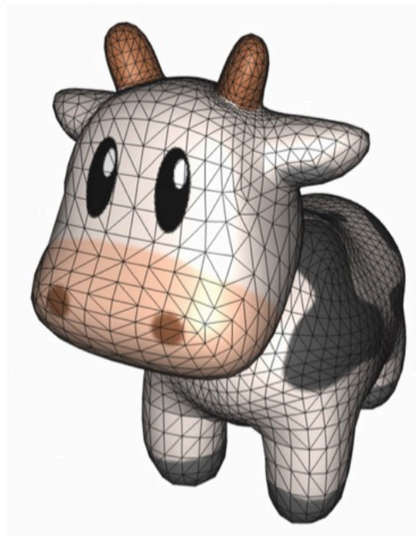
OBJ file

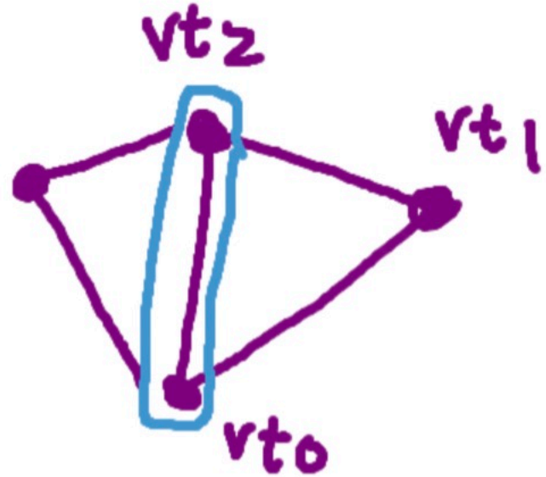
vt texture coordinates

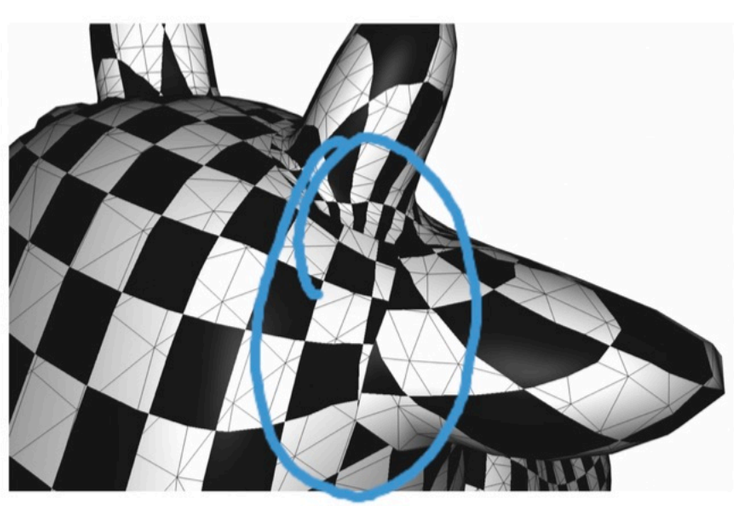
s t

u v







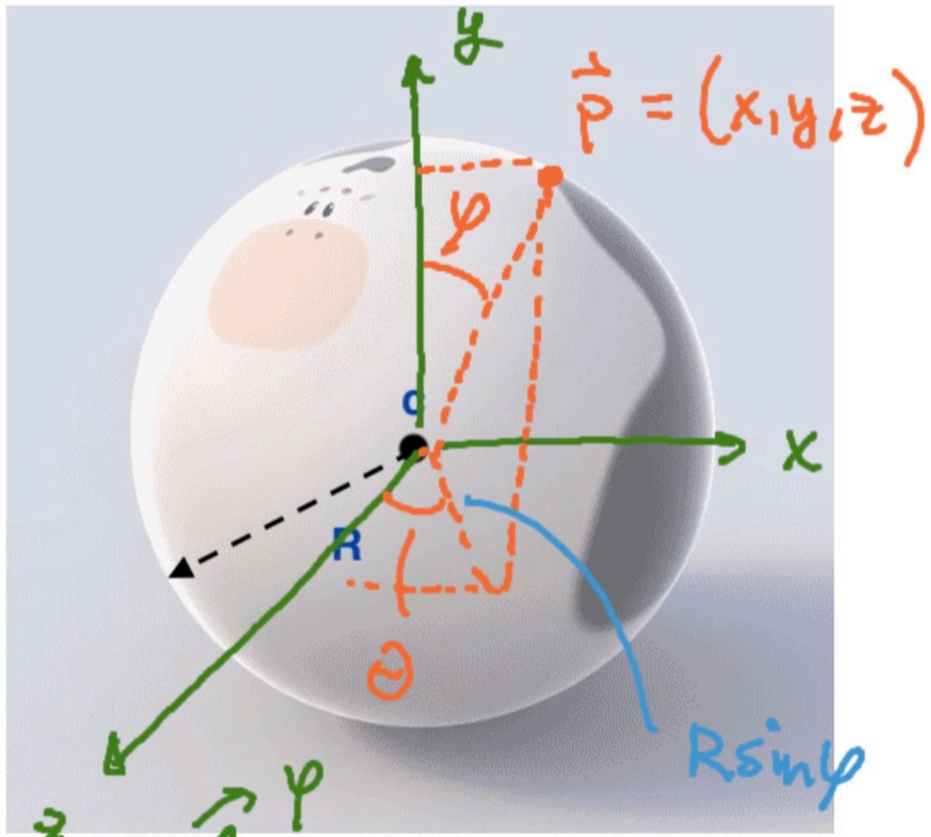


A special case: texture coordinates on a sphere.

Why? approximate lots of things like spheres!

WARNING: this will look a bit different than physics/math books.

(we are looking in $-z$-direction)



$\rightarrow y = R\cos\varphi$

$z = R\sin\varphi\cos\theta$

$x = R\sin\varphi\sin\theta$

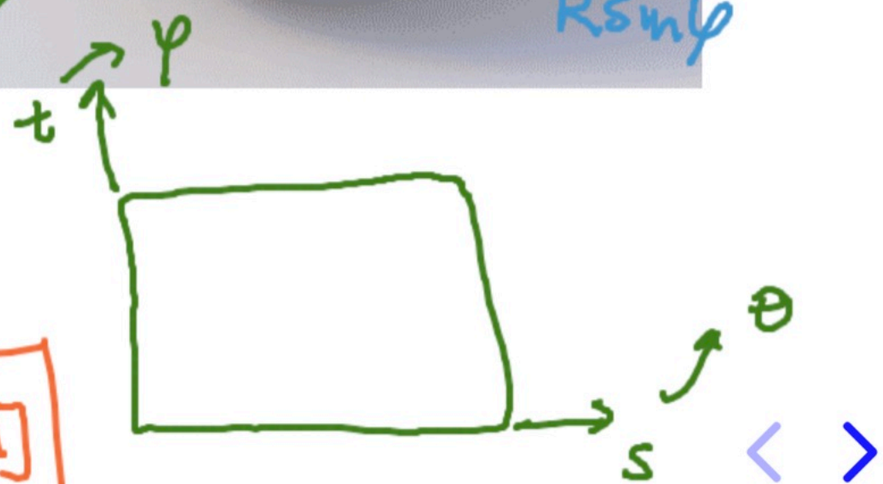
what are φ, θ given (x, y, z)?

$$\varphi = \arccos\left(\frac{y}{R}\right) \in [0, \pi] \rightarrow t = \frac{\varphi}{\pi}$$

$$\frac{x}{z} = \tan\theta \rightarrow \theta = \arctan\left(\frac{x}{z}\right)$$

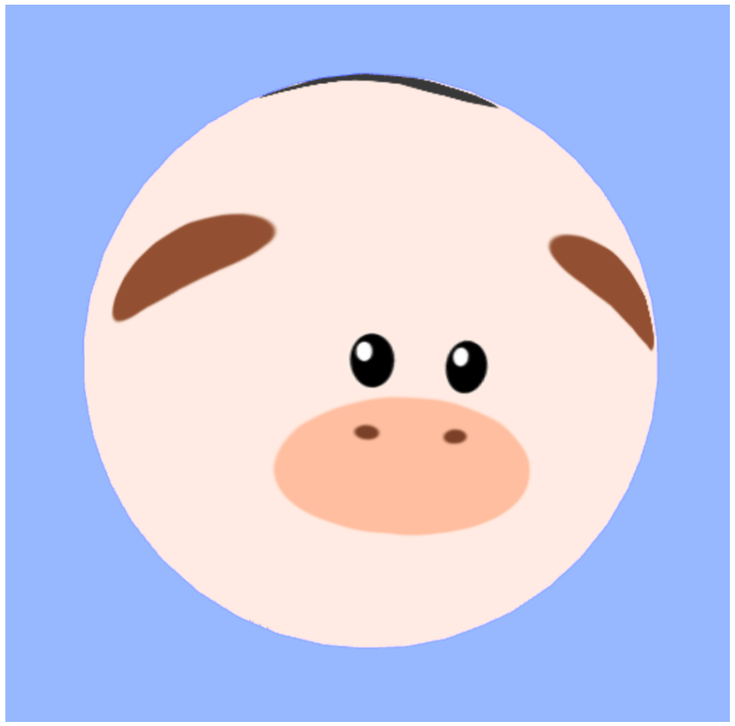
$\theta = \text{atan2}(x, z)$ $\uparrow$ in JS $[-\pi, \pi]$ $\rightarrow$

$$s = \frac{\theta + \pi}{2\pi} \in [0, 1]$$



Let's implement some of these ingredients!

Click on `exercise` in row for today's class. If you have `git` and `VS Code` set up locally, try working on your computer instead of a Codespace.



- Exercise 1: calculate texture coordinates for a point on the sphere.
- Exercise 2: use `ixn.textureCoords` to sample texture.
- Exercise 3: soon.

We can add callbacks to control the rotation of our model.

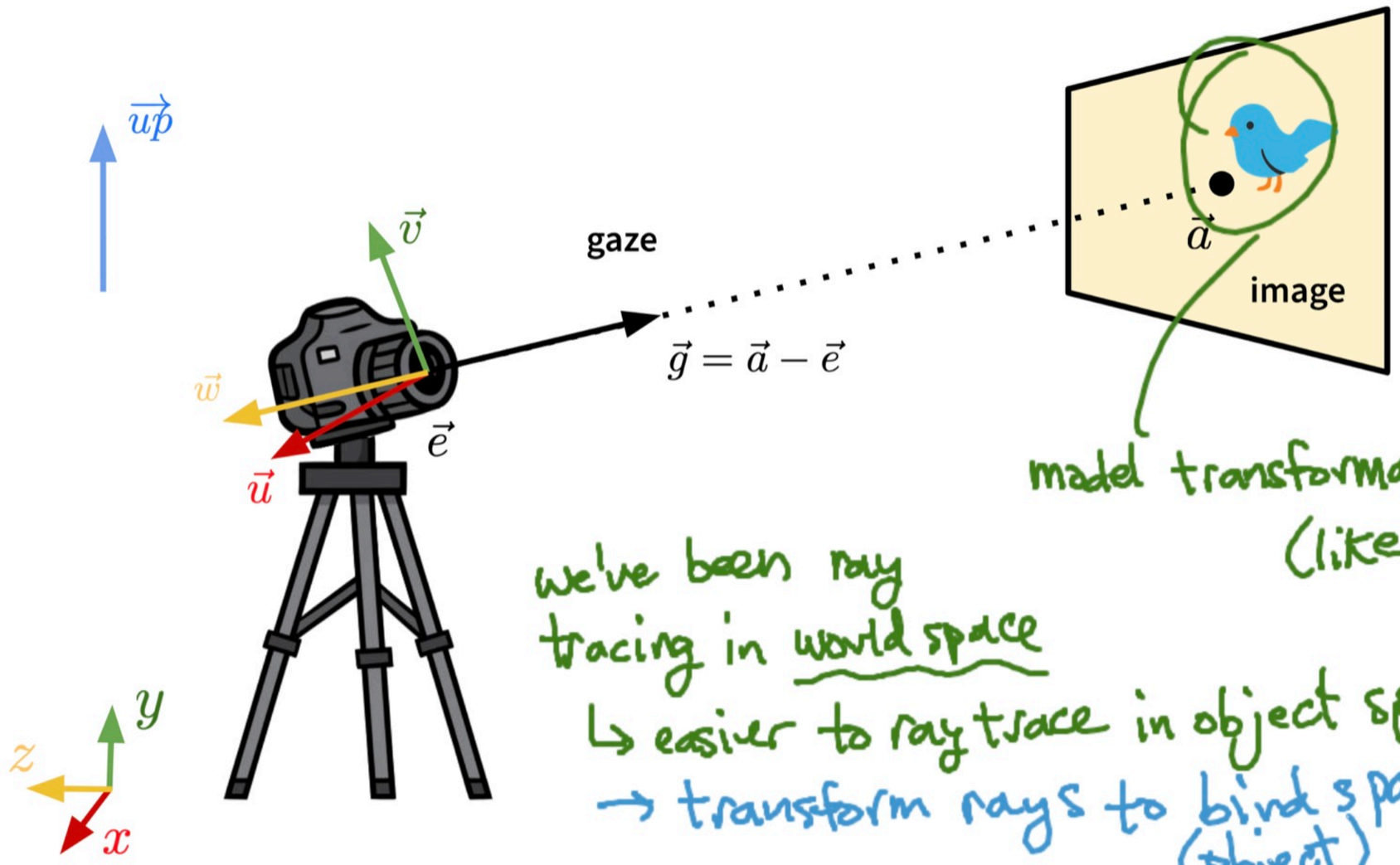
```
let dragging; // whether we are in dragging mode
let mouseX, mouseY; // current mouse position
canvas.addEventListener("mousedown", (event) => {
  dragging = true;
  mouseX = event.pageX; // save current mouse position
  mouseY = event.pageY;
});

canvas.addEventListener("mouseup", () => {
  dragging = false;
});

canvas.addEventListener("mousemove", (event) => {
  if (!dragging) return;

  // calculate angles/distance to rotate/translate from current/previous mouse position
  // ...
  render(); // re-render
  mouseX = event.pageX;
  mouseY = event.pageY;
});

canvas.addEventListener("wheel", (event) => {
  event.preventDefault();
  // perhaps move the eye to scroll in/out
  // ...
  render(); // re-render
});
```

How should we ray trace if our model has a transformation? (Exercise 3)

assume: looking in $-z$-direction $\vec{w} = \begin{bmatrix} 0 \\ 0 \\ +1 \end{bmatrix}$

$\vec{u} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$\vec{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$



modeled transformation M (like in Lab 4)

we've been ray tracing in world space

↳ easier to ray trace in object space

→ transform rays to blind space (object)

M: transf. object → world

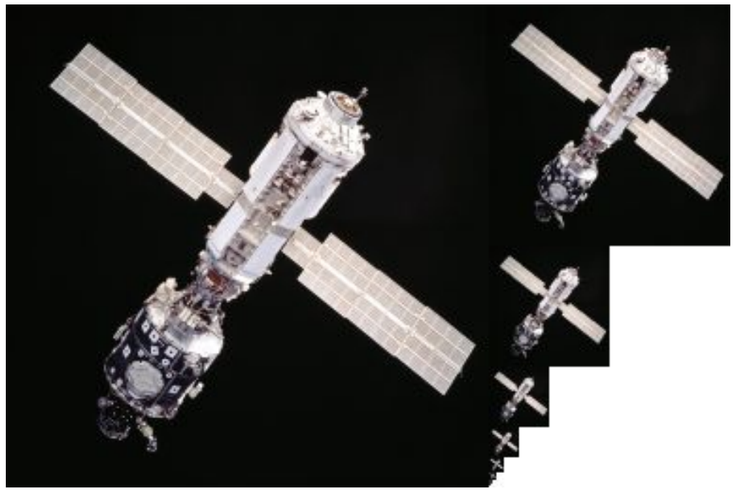
M^{-1}: transf. world → object

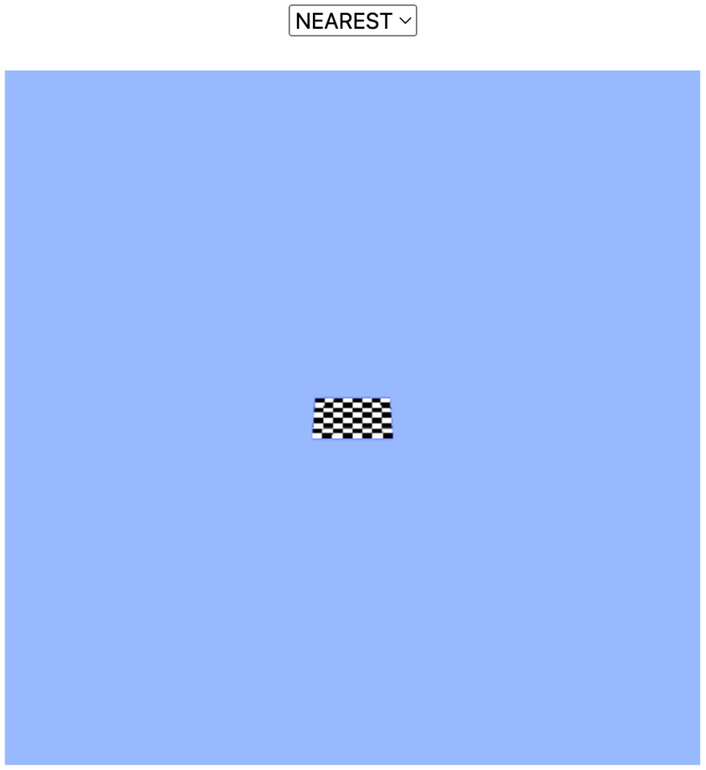
$$\vec{x}_{obj} = \underbrace{M^{-1}\vec{o}}_{\text{transf. point}} + \underbrace{M^{-1}\vec{r}}_{\text{transf. vector}} t$$

Texturing artifacts influenced by texture image resolution and distance to surface.

- **Magnification:** many screen pixels map to single texel: looks blocky. Average of a few texels?
- **Minification** single screen pixel covers many texels: shimmering effect. Which one to pick?

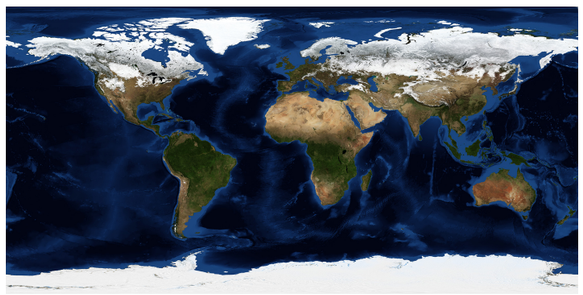
And how to make lookup efficient? **mipmaps**

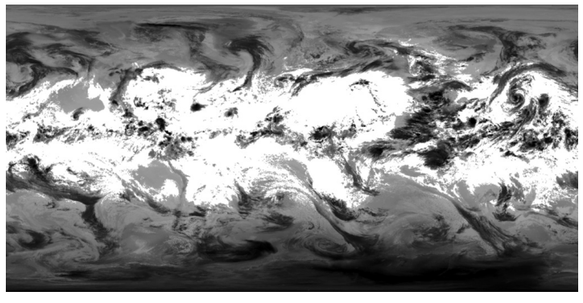


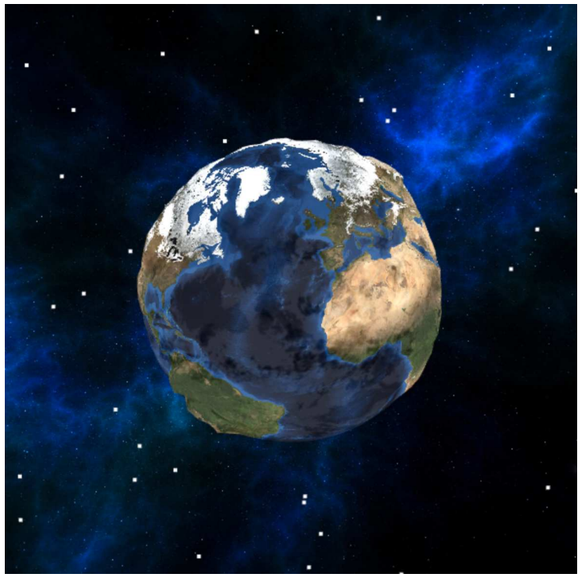


Other properties to modify with textures?

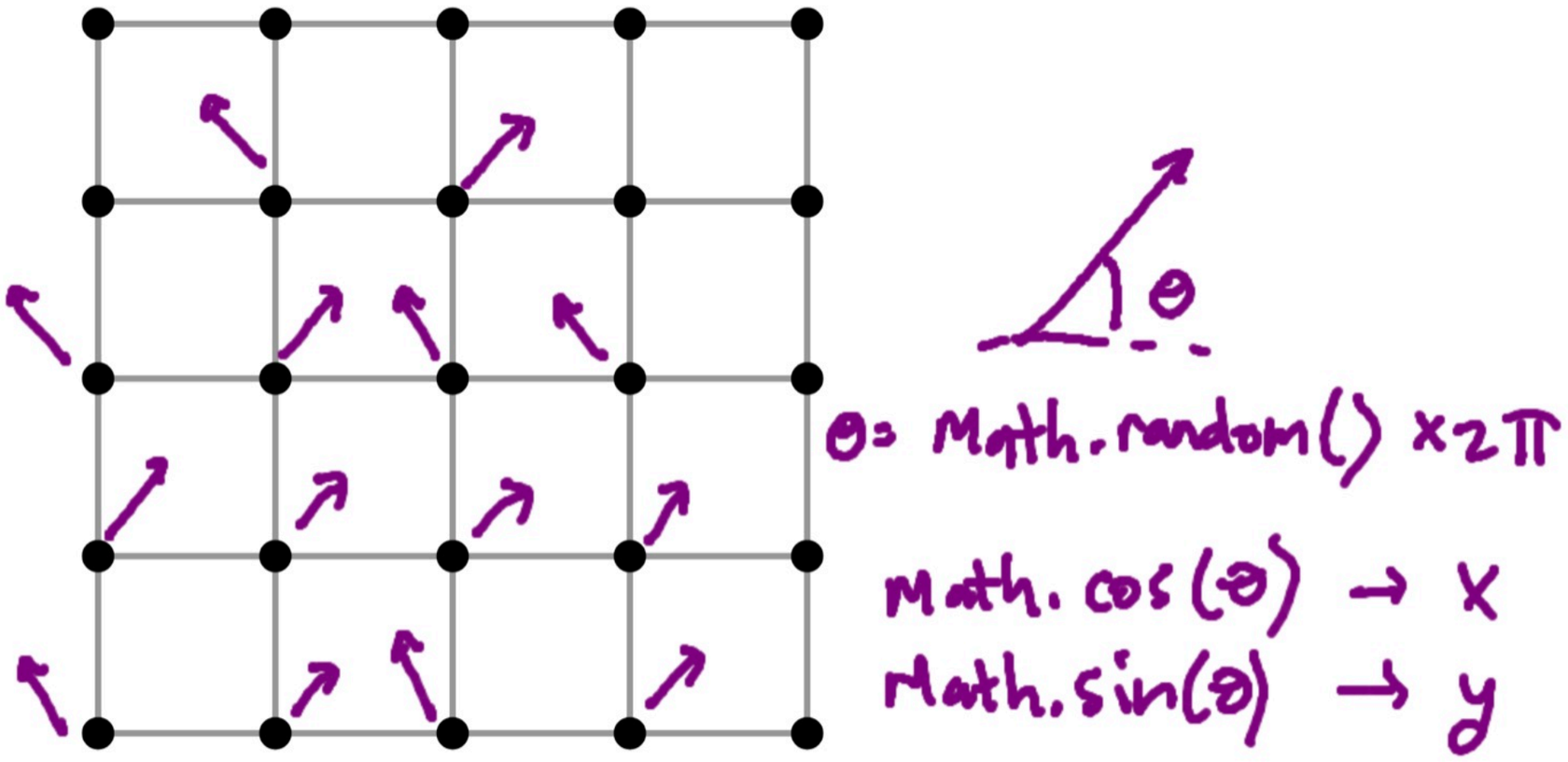
$$I = c_a k_m + c_l k_m \max\left(0, \vec{n} \cdot \vec{l}\right) + c_l k_s \max(0, \vec{v} \cdot \vec{r})^p$$







Implementing the Perlin noise algorithm (Part 1).



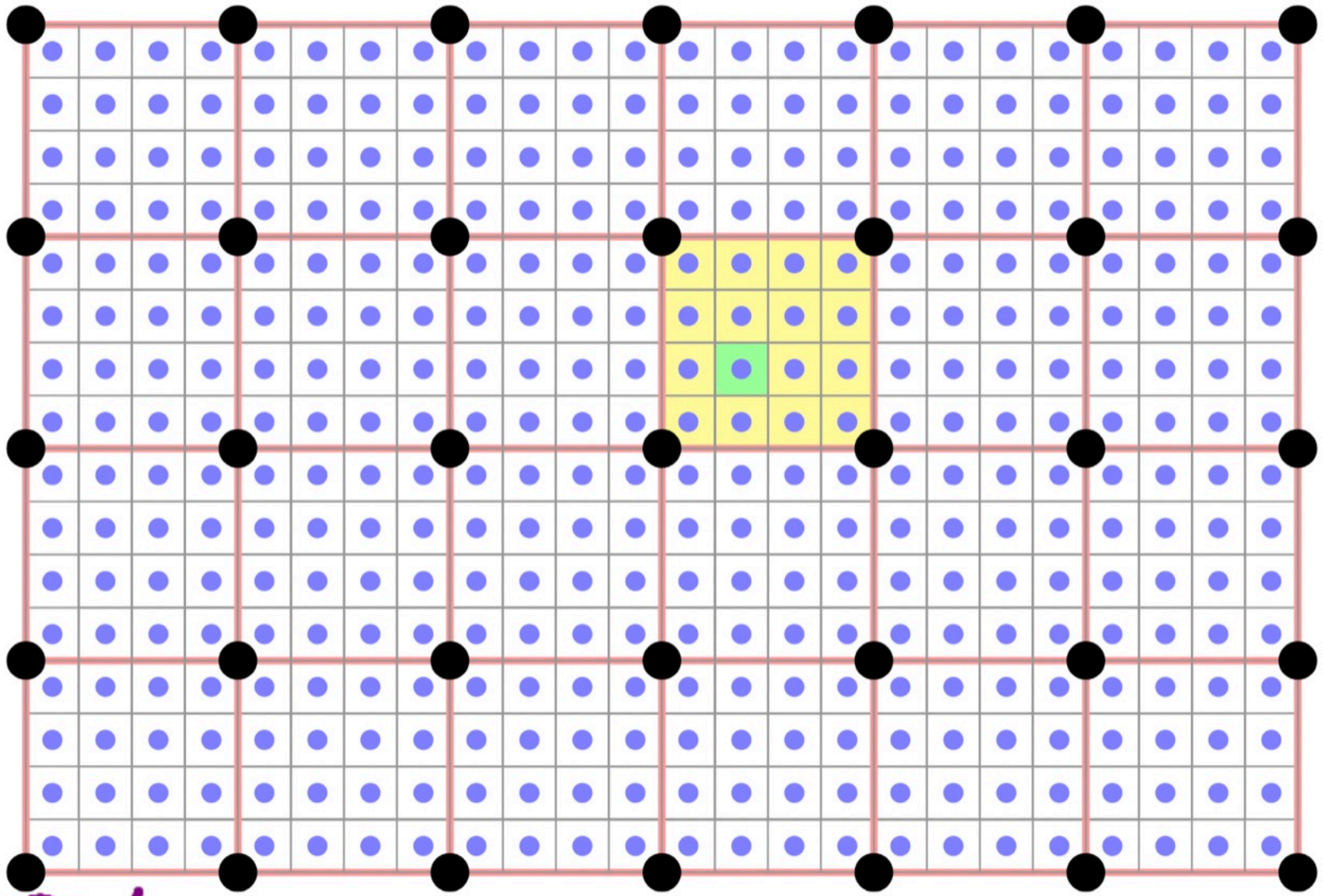
Define random unit vectors at each node of a *background* grid.

Implementing the Perlin noise algorithm (Part 2).

$$x = \frac{(i + 0.5)}{l_x}$$

$$l_x = \frac{npx}{ngx}$$

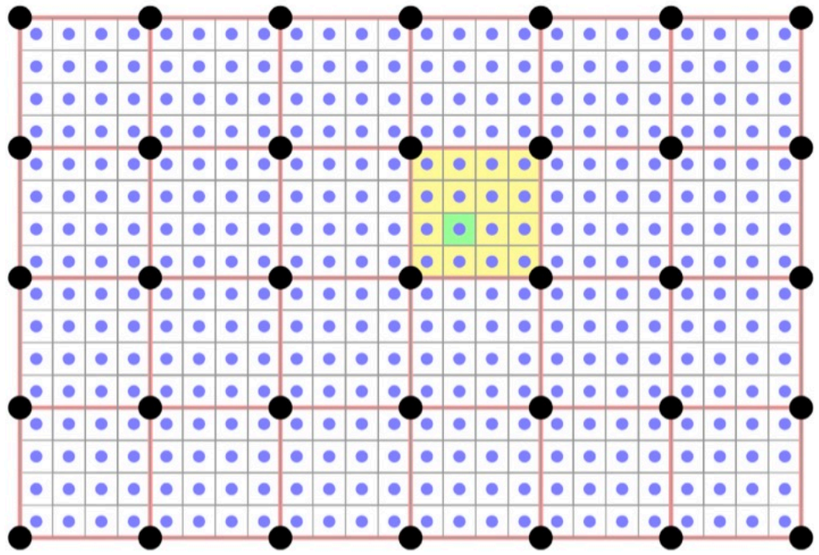
$$y = \frac{(j + 0.5)}{l_y} \quad l_y = npy/ngy$$

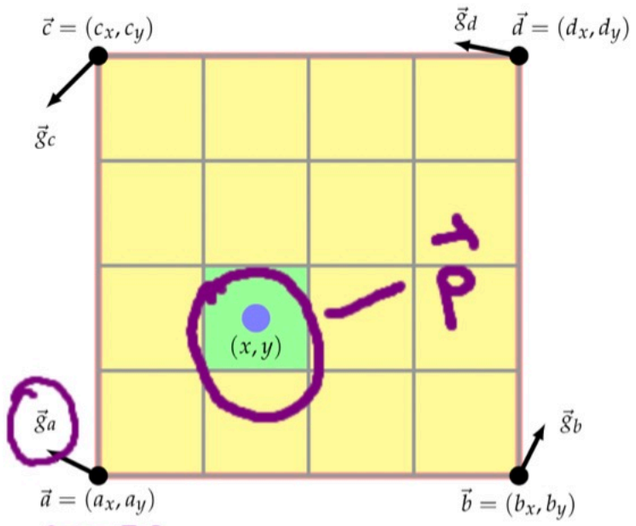


What are the coordinates of a pixel `(i, j)` relative to the grid system (i.e. # of grid cells in horizontal/vertical directions)?

Implementing the Perlin noise algorithm (Part 3).

Calculate the intensity (weight) that will eventually be used to interpolate two colors (blue and white for clouds).





calculate:

$d_a = (\vec{p} - \vec{a}) \cdot \vec{g}_a$

$d_b = (\vec{p} - \vec{b}) \cdot \vec{g}_b$

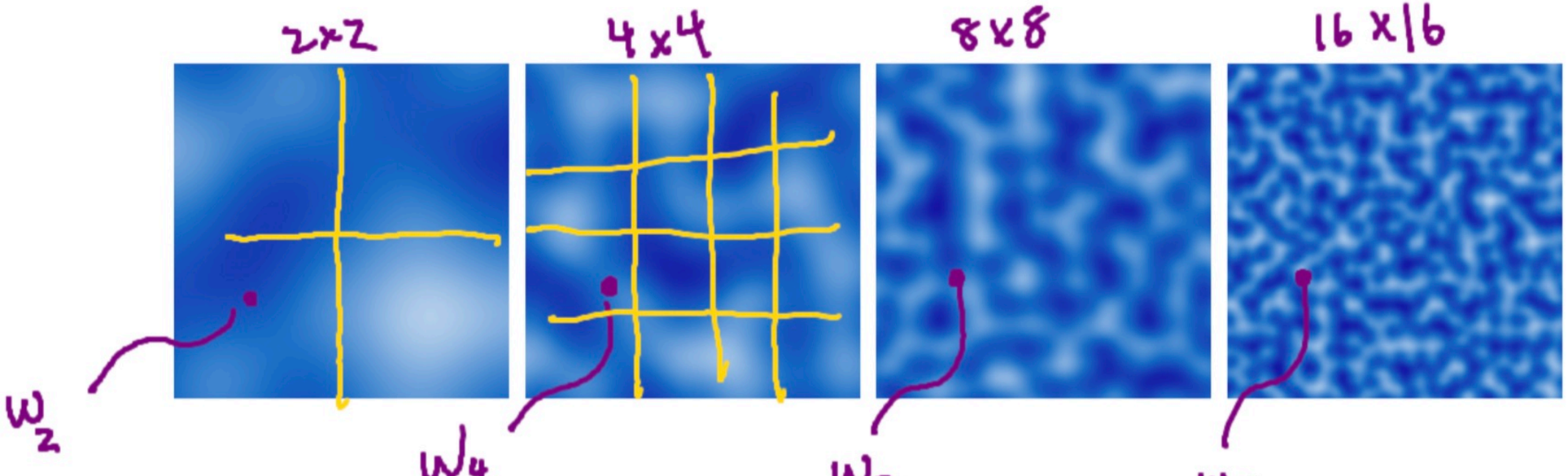
⋮

d_c

d_d

then interpolate.

Implementing the Perlin noise algorithm (Part 4).



$a_0 = 1$ $a_1 = 0.5$ $a_2 = 0.25$ $a_3 = 0.125$

$$w = \sum_k a_k w_k$$

k ← grid index

Summary

- Calculate or retrieve texture coordinates.
- Texture can define color, normal, displacement, specular coefficient, anything!
- Displacement mapping is hard to do with ray tracing, much more do-able with rasterization.
- If you have a `.glb` file, extract texture images: https://products.aspose.app/3d/extractor/glb.

$$I = c_a k_m + c_l k_m \max\left(0, \vec{n} \cdot \vec{l}\right) + c_l k_s \max(0, \vec{v} \cdot \vec{r})^p$$