Other Effects

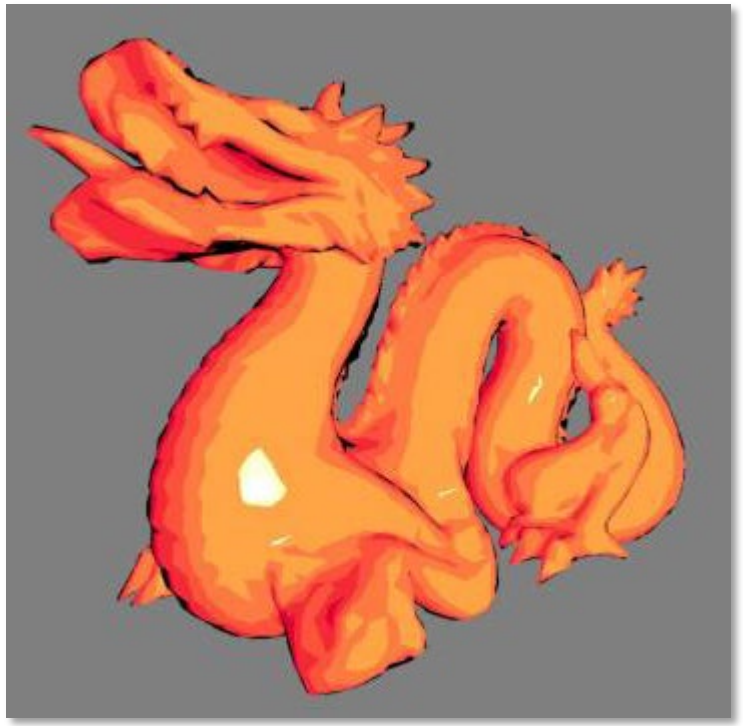
Other effects

- Non-Photorealistic Shading
- Outlines & edge detection
- Emissive Textures
- Fog

Non-Photorealistic Shading

Non-photorealistic shading

- Our BRDFs assume we're trying to mimic real world lighting
- But what if we don't want that?

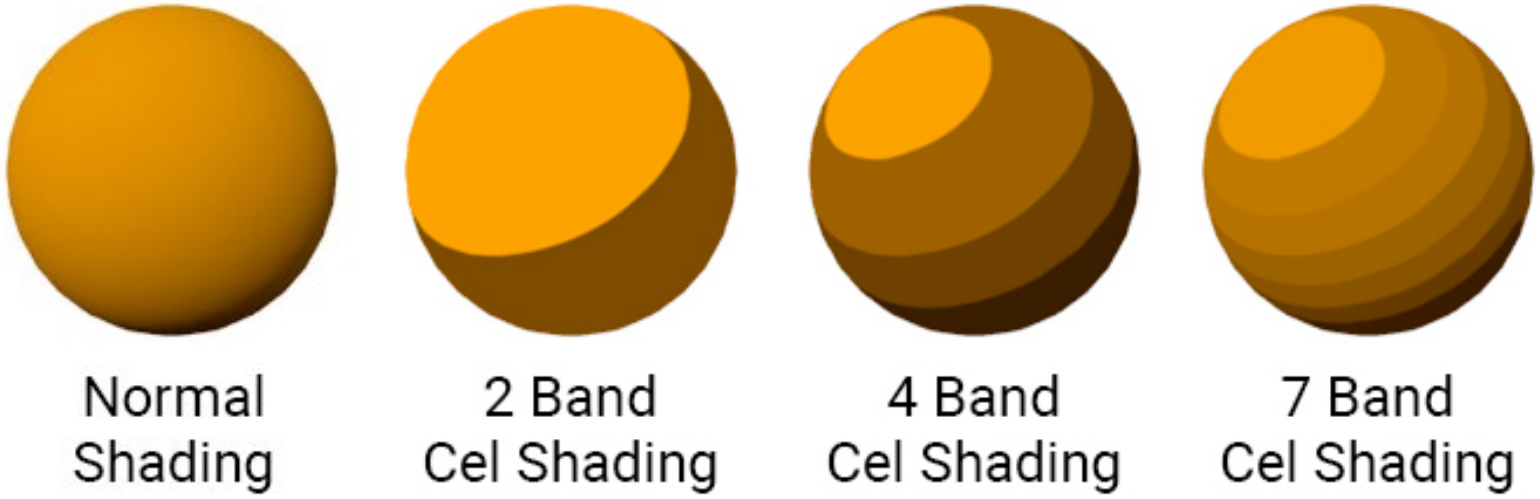


Cel shading

- Short for “celluloid”
  - Clear sheets on which cartoons were painted
  - These sheets were often called *cels*
- Sometimes called “Toon Shading”
- Often combined with an outline effect to look like an actual cartoon drawing

Cel shading goal

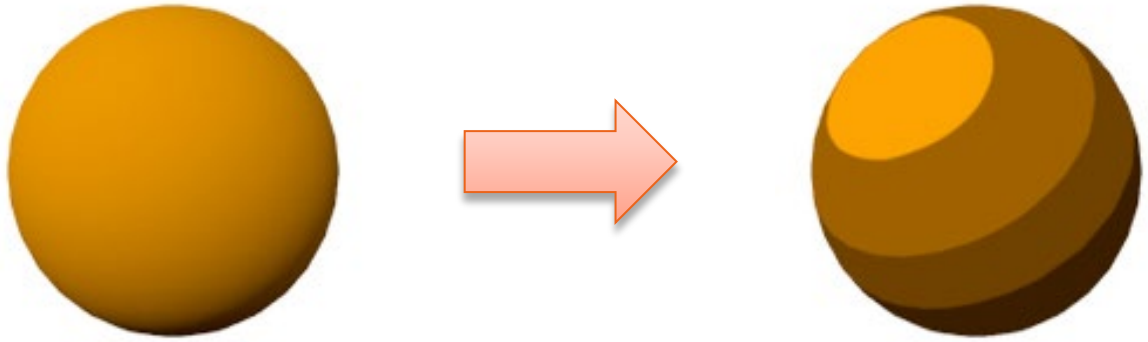
- Bands of colors instead of smooth gradients



Cel shading basics

- Diffuse calculation returns 0-1 shading value

- Tweak that value to create bands of colors
  - Instead of any number between 0 – 1
  - Use only a small set of diffuse values
  - For example: 0.0, 0.4, 0.75, 1.0



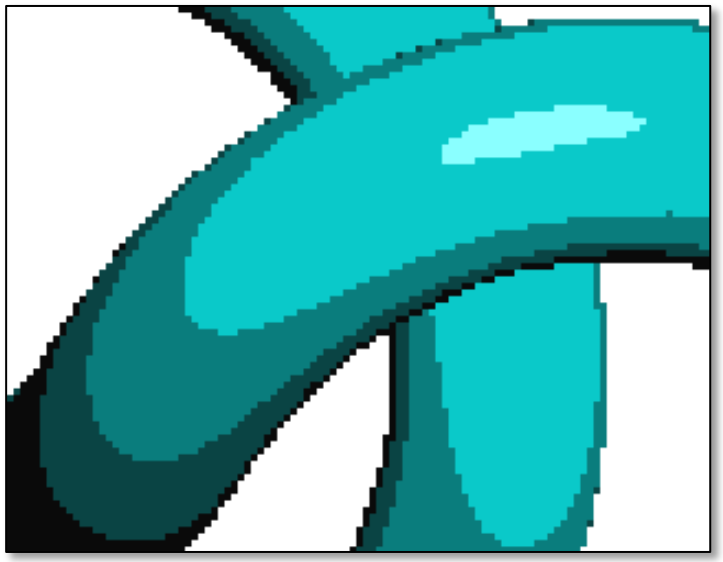
Implementing cel shading

- How do we make “color bands”? If statements?

```
if( diff > 0.9 )
   diff = 0.9;
else if( diff > 0.6 )
   diff = 0.6;
else
   diff = 0;
```

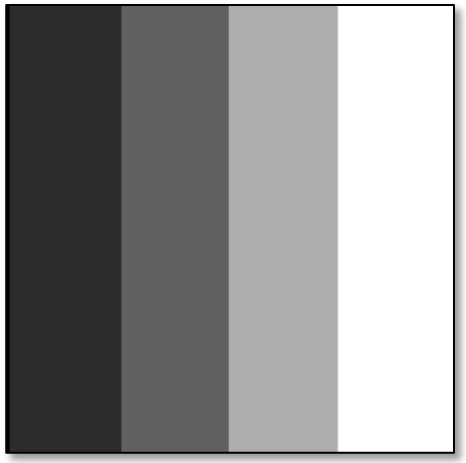
- Not great…

- Better solution: A ramp texture

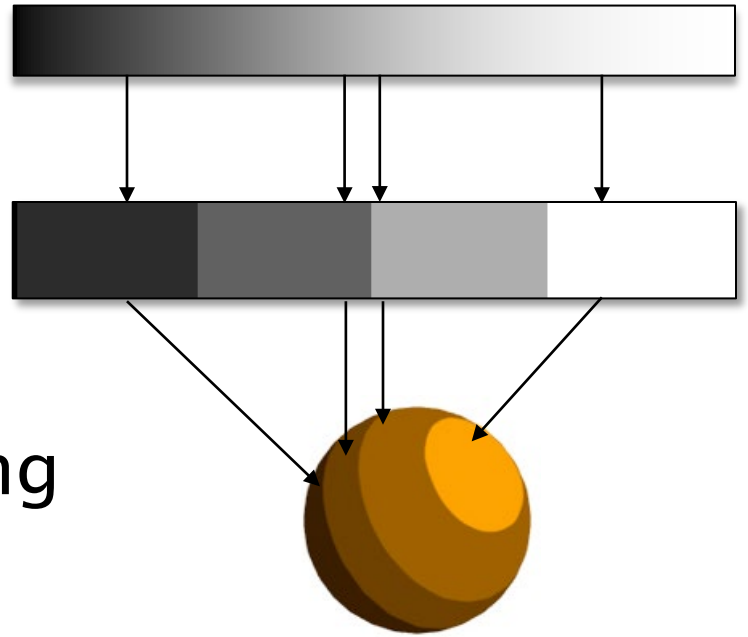


Aliased color bands
using if statements

Ramp texture



- A texture with color bands
  - Tweakable by an artist without code
  - Supports any number of bands
  - Can be colored or greyscale

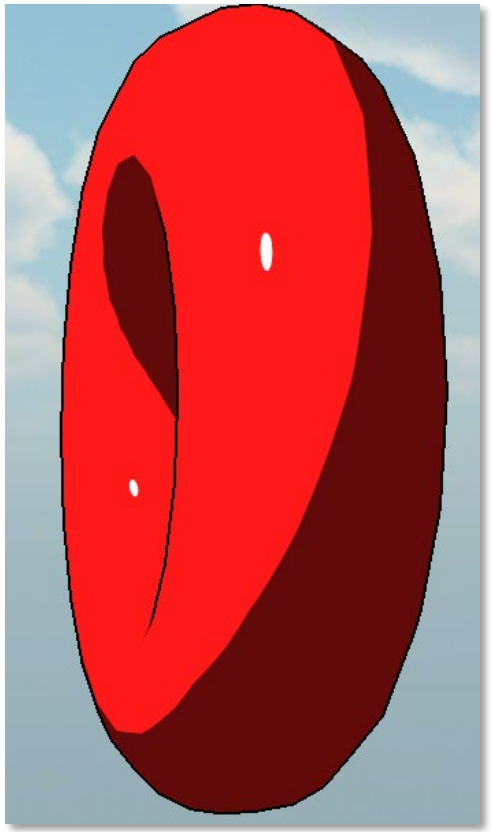


- 1. Calculate standard diffuse lighting
- 2. Use #1 to look up color band
  - rampUV = float2(diffuse, 0);
- 3. Use #2 as actual diffuse term for shading

Outlines & Edge Detection

Outlines & edge detection

- Often used in conjunction with cel shading to look cartoony
- A more complex effect than cel shading
  - Not just an addition to our existing shader
- Problems
  - A pixel doesn't know if it's "on the edge"
  - Outline is literally outside the geometry
  - Outline should be a fixed width



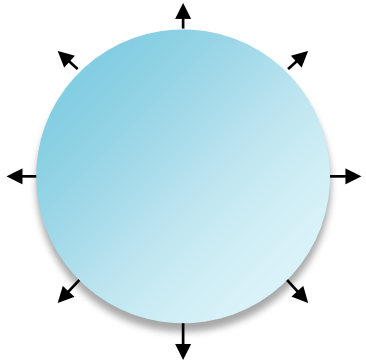
Outline implementations

- Many implementations exist, each with different results
- Inside-out geometry
- Sobel filter (post process)
- Silhouette post process
- Normal/Depth compare post process

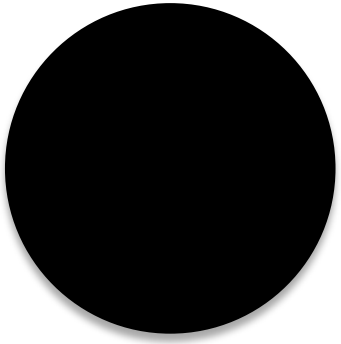
Inside-out geometry

- Re-render a mesh inside out
- Extrude each vertex along its normal in a special vertex shader
- Use black as the pixel color

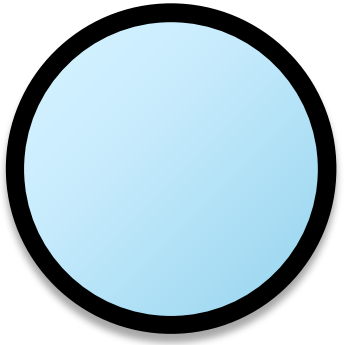
Original sphere



Inside-out sphere



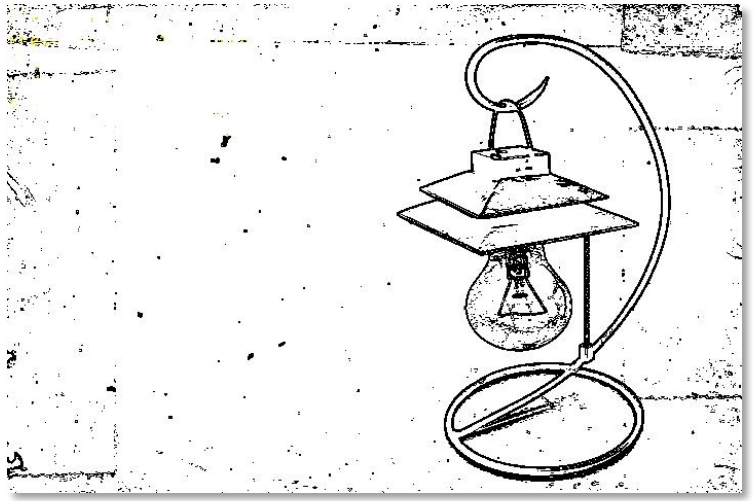
Now slightly larger & black



Original + inside out

Sobel filter

- Post process for edge detection
  - Compares neighboring colors and weights them
- Pros
  - Only requires final image
  - Great at finding obvious edges
- Cons
  - False positives
  - Not great unless colors are flat
  - Noisy textures = noisy edges



Sobel implementation

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Sobel weights

- Sample neighbors & weight
  - `horizTotal += neighbor[x,y] * horizWeight[x,y];`
  - `vertTotal += neighbor[x,y] * vertWeight[x,y];`
- Calculate magnitude and add components
  - `mag = sqrt( horizTotal * horizTotal + vertTotal * vertTotal );`
  - `sobel = saturate(mag.x + mag.y + mag.z);`
- Interpolate between pixel color and edge color
  - `finalColor = lerp(pixelColor, edgeColor, sobel);`

Silhouette outlines

- Render a unique “silhouette ID” for each object
  - Can output as each pixel’s alpha value (if no transparency)
  - Allows for up to 256 different IDs
- Compare silhouette IDs in post process shader
- If any neighbors of a pixel have a different ID, this pixel is an edge

Silhouette results

- Pros
  - Very precise edges
- Cons
  - Limited number of IDs
  - No interior edges

Normal/Depth comparison

- Compare a pixel's normal & depth to its neighbors

- If normal changes rapidly → probably an edge
- If depth changes rapidly → probably an edge

- Requires the use of *multiple active render targets*
  - Output each pixel's normal and depth to separate targets
  - At the same time as colors

Normal/depth results

- Pros
  - Handles interior edges
- Cons
  - Can still miss some edges
  - More complex implementation

Emissive Textures

Emissive Textures

- Making an object look like it emits light
  - (It won't *actually* emit light, however)

Emitting light

- Remember the Rendering Equation?

$$\underbrace{L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}, \lambda, t)}_{\text{Outgoing light}} = \underbrace{L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}, \lambda, t)}_{\text{Emitted light}} + \underbrace{\int_\Omega f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o}, \lambda, t) L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}, \lambda, t)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}}_{\text{Incoming light}}$$

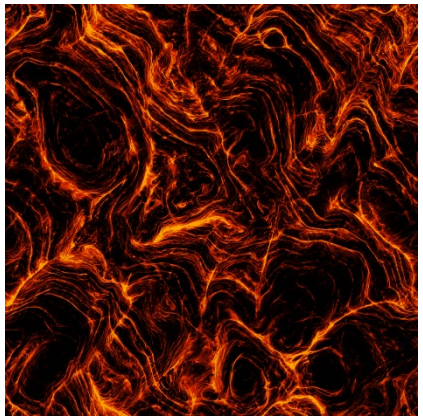
- Our lighting equations so far have dealt with *incoming light*
- If we want to “emit light”, we simply add it to the result

How much light is emitted?

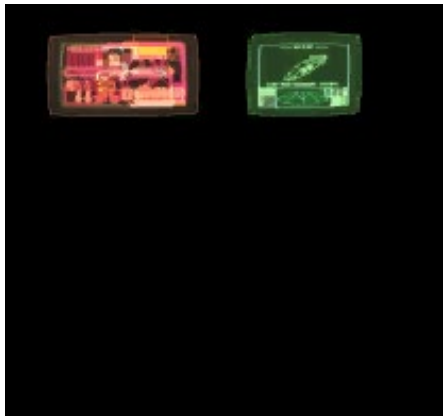
- Depends on the surface
- Light bulbs appear to exclusively emit light of a single color
- Lava has bright (emissive) and dark (non-emissive) areas
- An RGB keyboard emits lots of different colors

Emissive textures

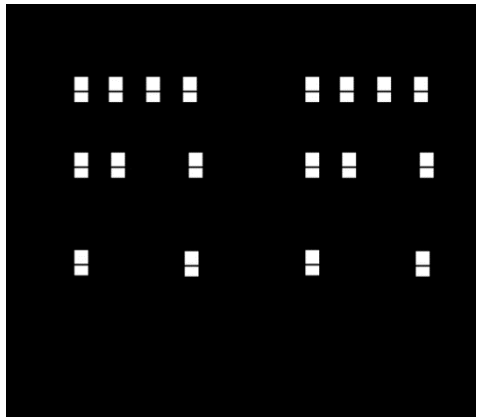
- Textures that represent emitted light color & intensity
  - Color = emitted light
  - Black = no emission



Lava



Computer screens

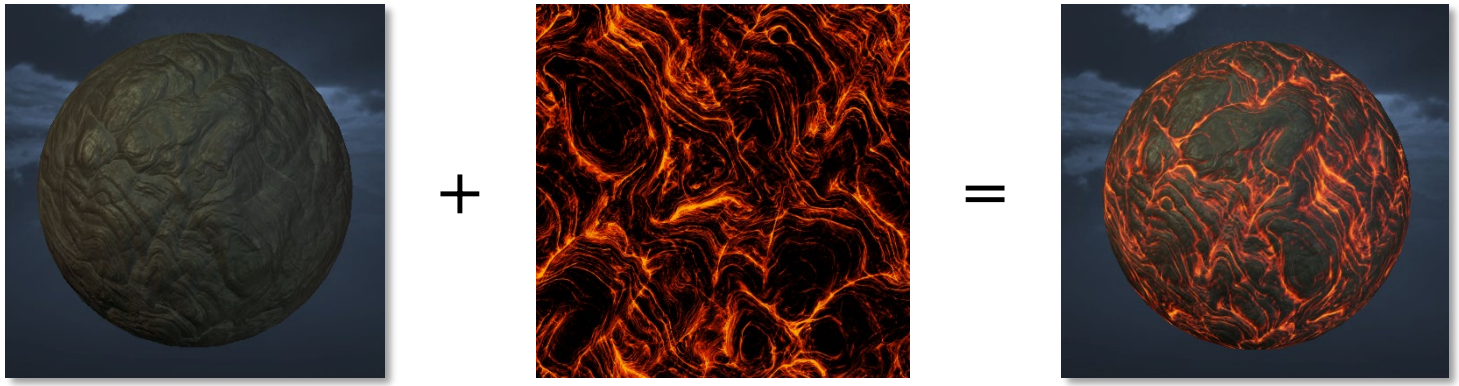


Apartment windows

Using an emissive texture

- Define an “Emissive Map” texture in your shader
- Sample and add the emissive color to the final light total

```
totalLight += emissiveColor;
```



Fog

Distance-based fog

- Objects fade to a solid color as they get farther away

Basic fog implementation

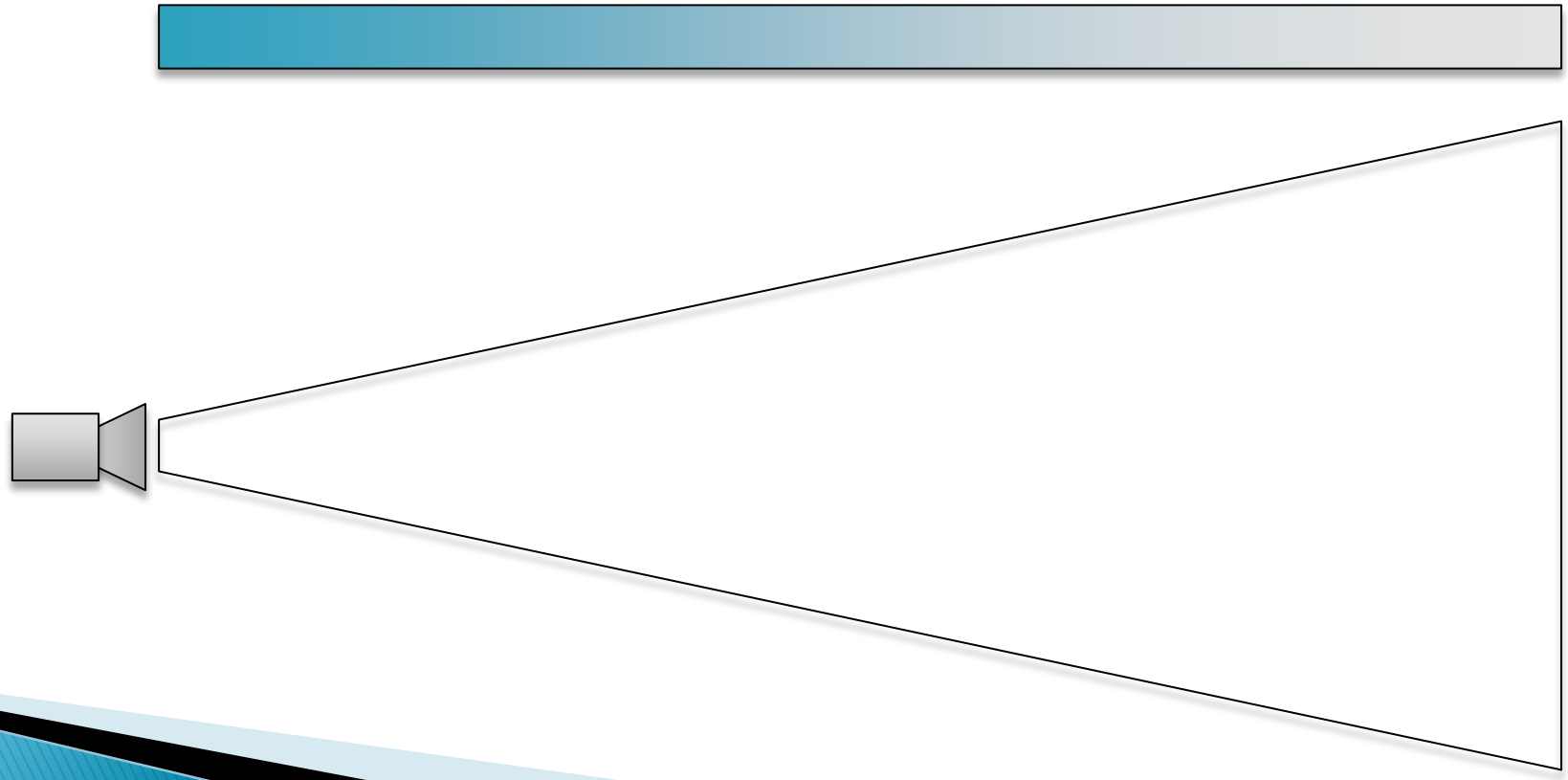
- Calculate the distance to the pixel
- Divide by far clip plane distance to get 0-1 value
- Use result to interpolate between pixel color and fog color

```
float dist = length(cameraPos - worldPos) / farClipDistance;
float3 finalColor = lerp(pixelColor, fogColor, dist);
```

- Works fine, but you might want more control
  - Where does fog “start”?
  - Where does fog “end”?

Fog example

- Fog results at various distances
- Objects fade to fog color (grey) as the move away



Fog parameters

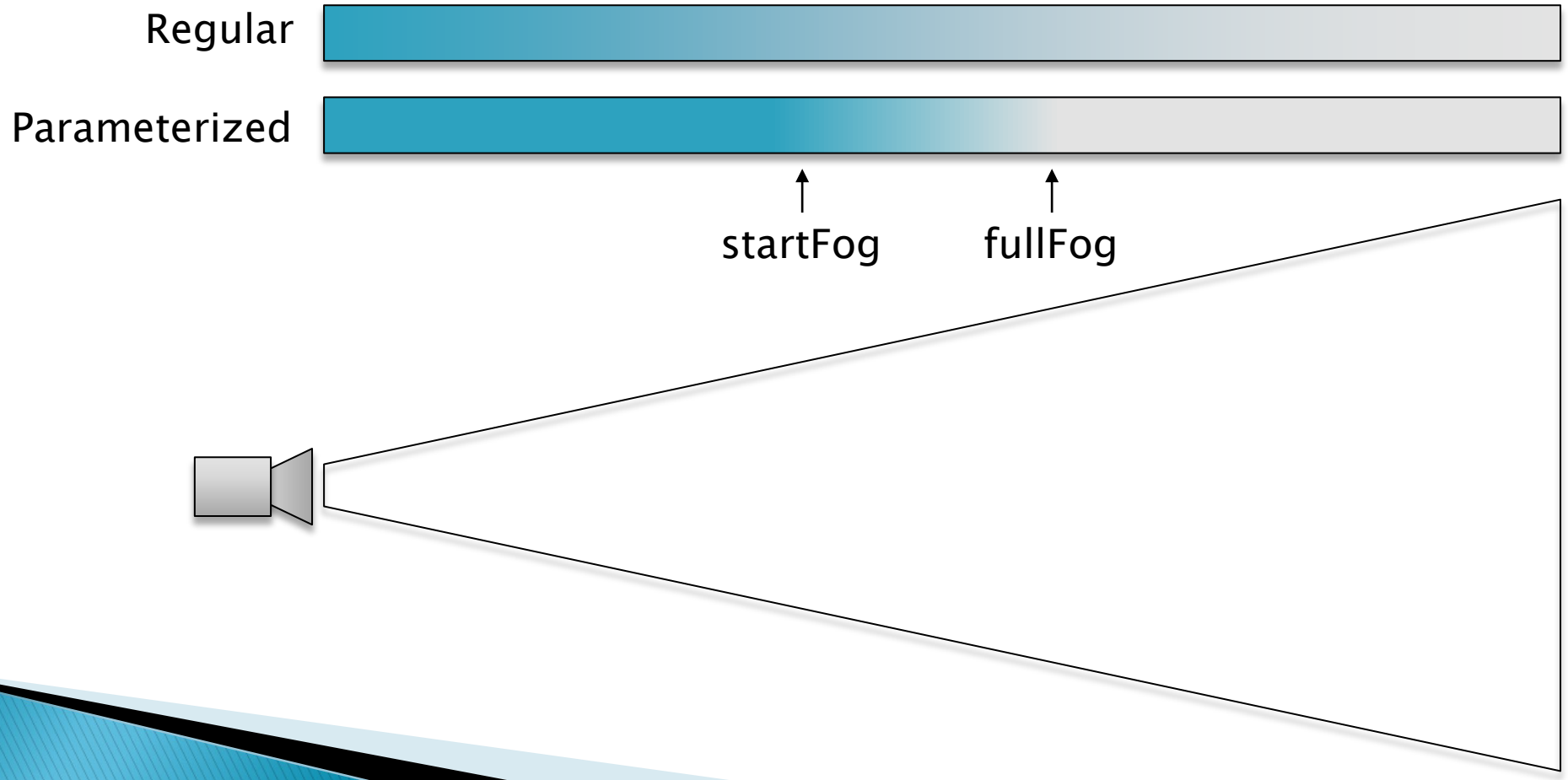
- Define a startFog and fullFog distances
  - startFog – where fog begins (anything closer is not fogged)
  - fullFog – where fog is max (anything farther is fully fogged)

- Use smoothstep(min, max, value)
  - Smoothly interpolates between min and max
  - Also clamps at min and max

```
float dist = length(cameraPos - worldPos);
float fog  = smoothstep(startFog, fullFog, dist);
float3 finalColor = lerp(pixelColor, fogColor, fog);
```

Parameterized fog example

- Fog results at various distances



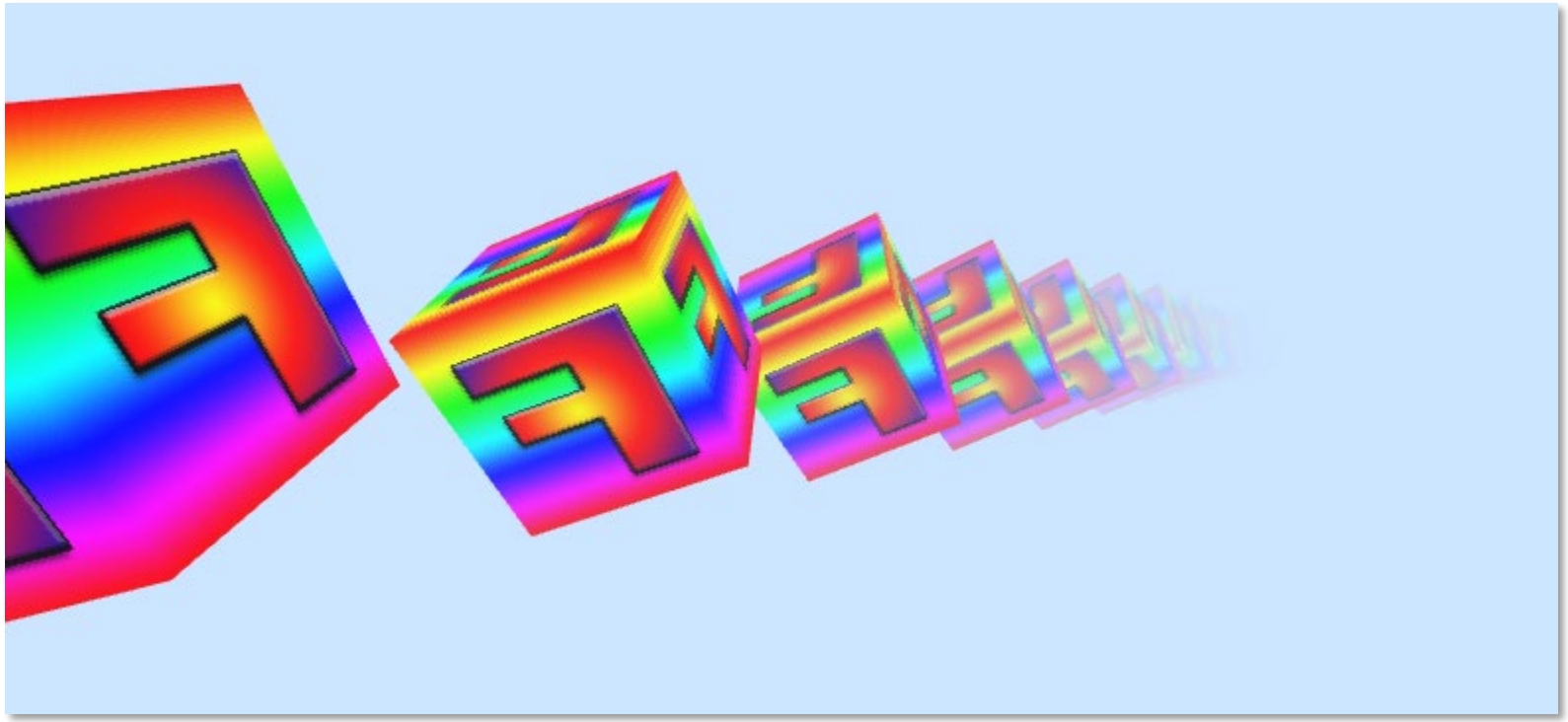
Exponential fog

- Fog in the real world isn't linear; it's exponential
- We might also want to define the fog's density

- We can handle both of these changes easily
  - Use exp() function, which applies the given exponent to *e*
  - Define a fog density between 0 and 1

```
float dist = length(cameraPos – worldPos);
float fog  = exp(-dist * fogDensity)
float3 finalColor = lerp(pixelColor, fogColor, fog);
```

Exponential fog example



Height-based fog

- Can also apply more fog based on Y position
  - Looks like objects rise above/through the fog