TA 6

- Texturing!
- Texture Mapping
  - Explicit UV Mapping
  - Projection Mapping
- Texture Sampling

Texturing

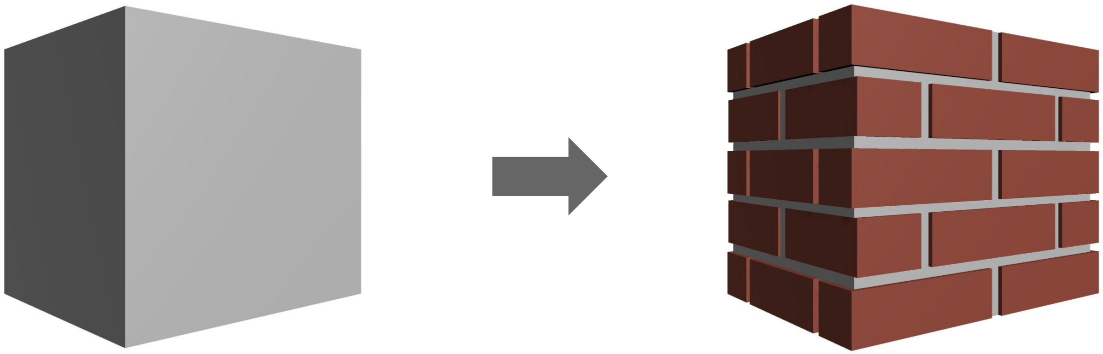
Computer Graphics 2020

Textures

- So far we have learned about using polygons, colors and lights to generate the look of all the objects in the scene
- What if we want to add more detail, for example - to create a brick wall?

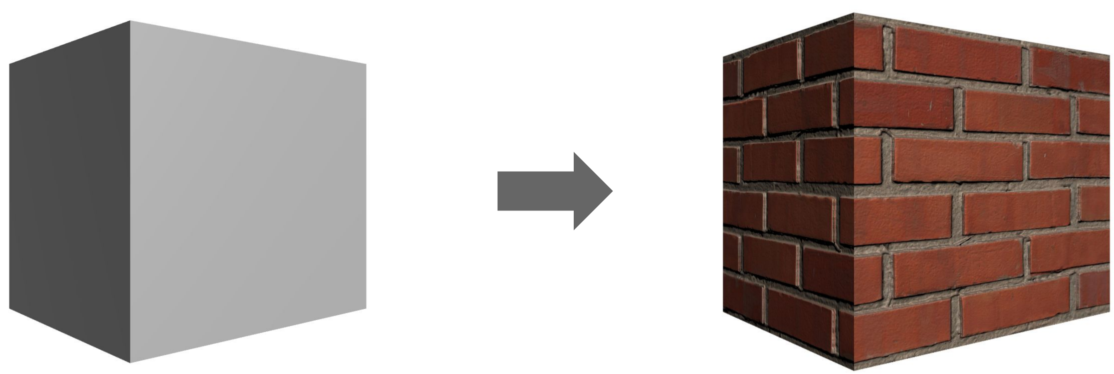
Textures

- We can add geometry to represent bricks
- This is inefficient - we need much more vertices and faces to describe the same cube!
- Not very realistic



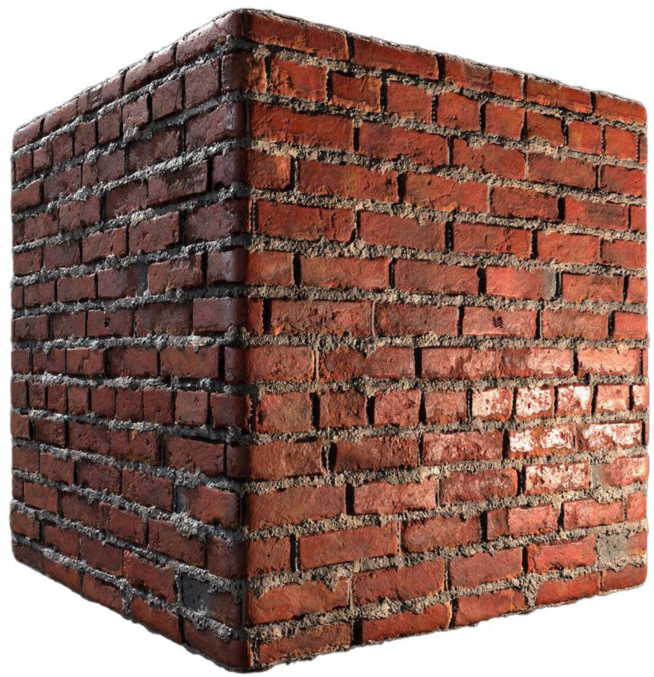
Textures

- A better way? Textures!
- Textures can add high-frequency details without changing the geometry
- Can be very efficient



Textures

- Using textures we can potentially achieve extremely realistic results



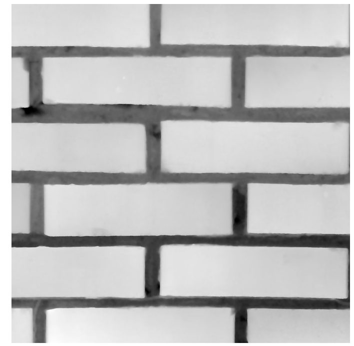
Textures

- ***Textures*** express material attributes of a surface without affecting the underlying topology of the geometry
- Attributes can include the color of the surface, how reflective it is, and many more properties

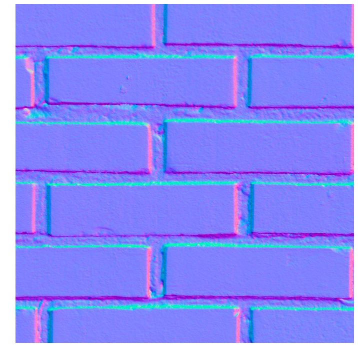
Some Types of Textures

Albedo/Color Map

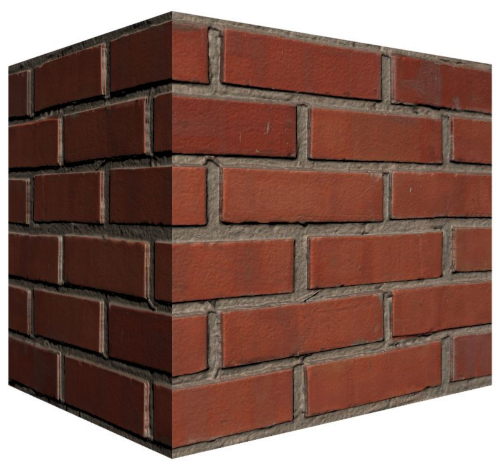
Specular Map



Height Map



Normal Map

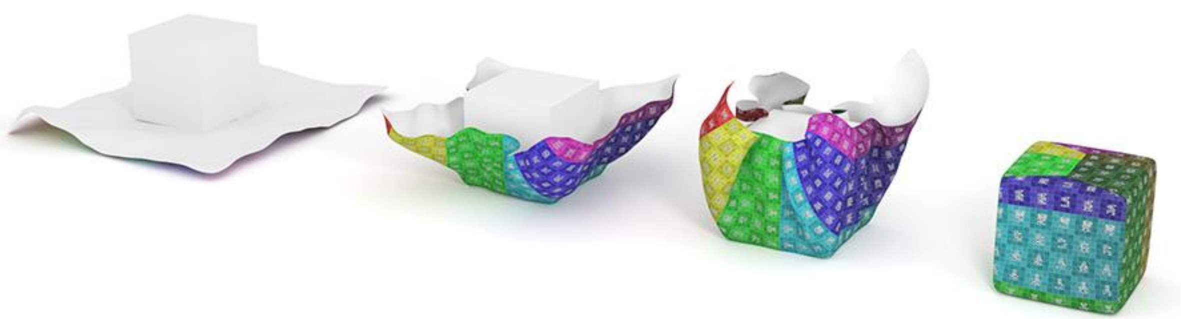


Textures

- Generally when talking about textures we mean either a 2D images or some 2D function
- A texture pixel is called a ***Texel***, to differentiate it from screen pixels
- We can also use 3D Textures - we will learn about them later

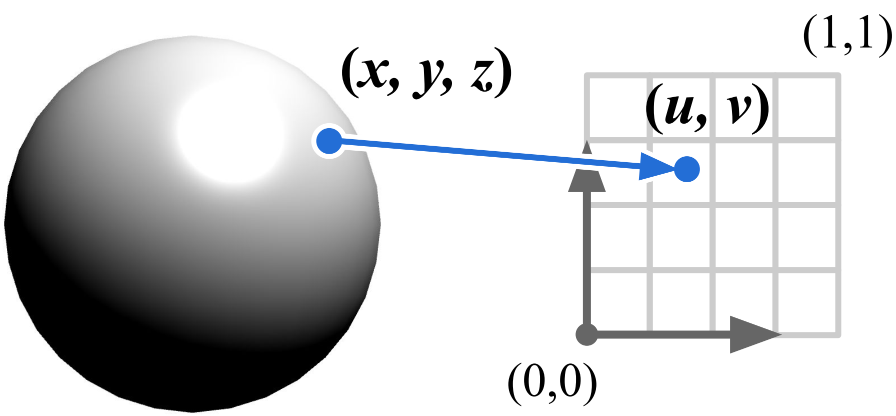
Texturing

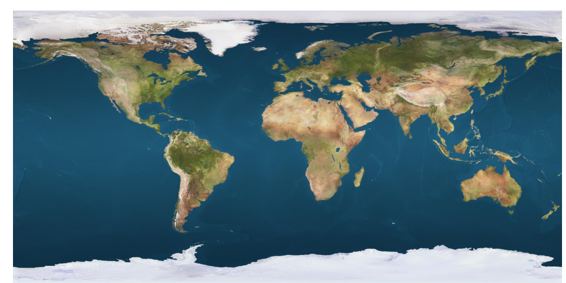
- ***Texturing*** is the process of taking a 2D texture and mapping it onto the surface of a 3D object in the scene
- It involves 2 parts: *Mapping* and *Sampling*



Texture Mapping

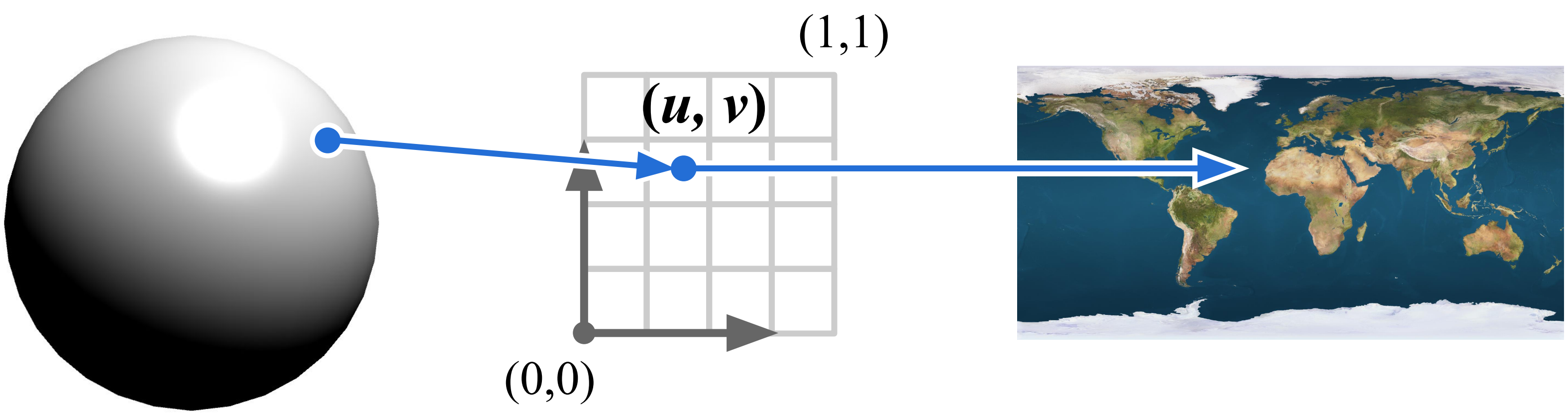
1. ***Mapping*** - use some function $f: \mathbb{R}^3 \rightarrow [0,1]^2$ to map 3D object coordinates (x, y, z) to 2D texture coordinates (u, v)





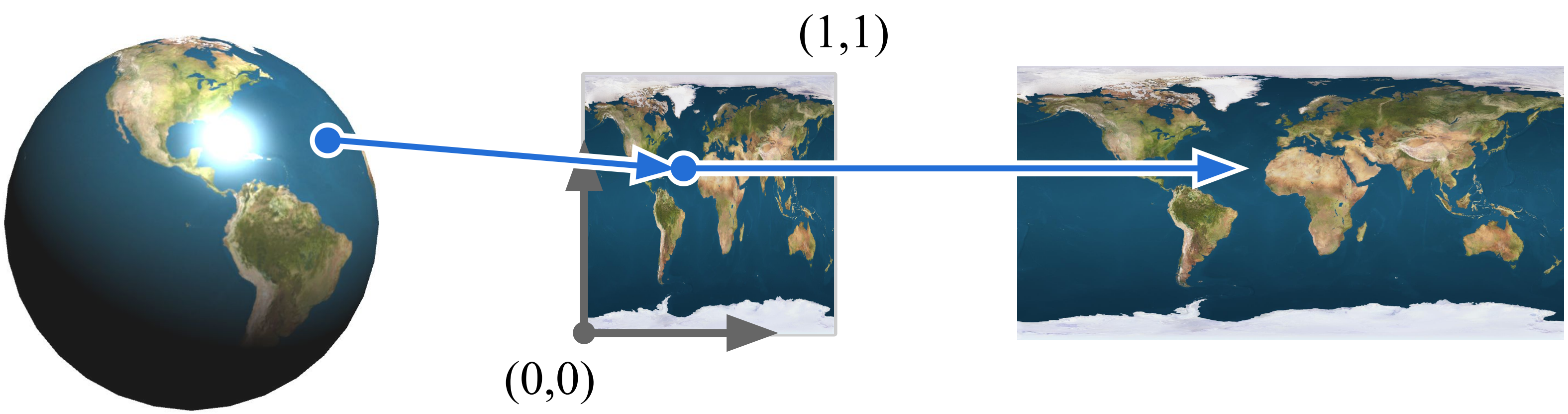
Texture Mapping

2. ***Sampling*** - Sample the texture at the coordinates (u, v) to obtain a value (color or some other attribute)



Texture Mapping

2. ***Sampling*** - Sample the texture at the coordinates (u, v) to obtain a value (color or some other attribute)



Texture Mapping

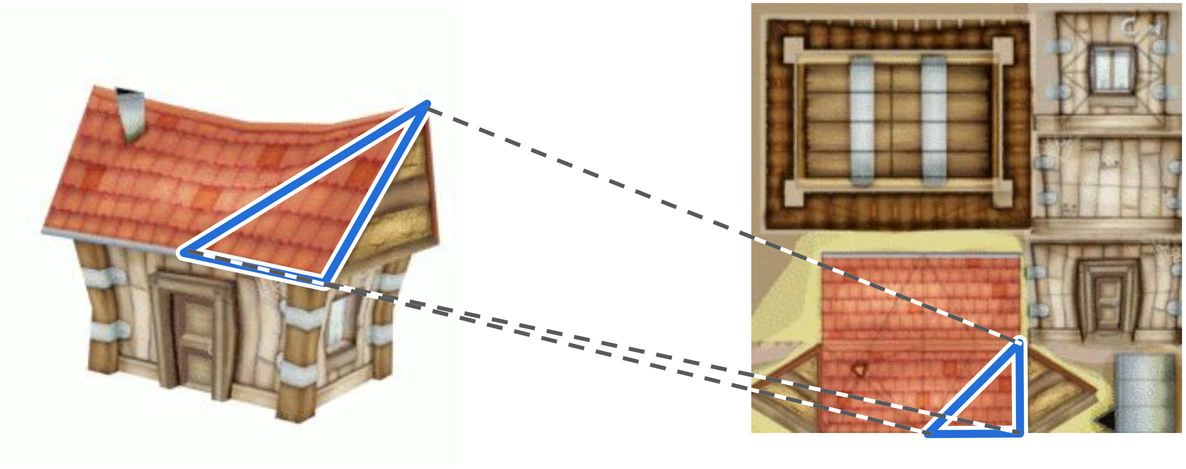
Computer Graphics 2020

Texture Mapping

- ***Texture Mapping*** or ***UV Mapping*** is the finding some function $f : \mathbb{R}^3 \rightarrow [0,1]^2$ to map 3D object-space coordinates (x, y, z) to 2D texture coordinates (u, v)
- We can either find some mathematical function f to do this for us, or explicitly define a mapping between points on our mesh and points on our texture

Explicit UV Mapping

- We can explicitly store (u, v) texture coordinates for each vertex, then interpolate between them at each fragment

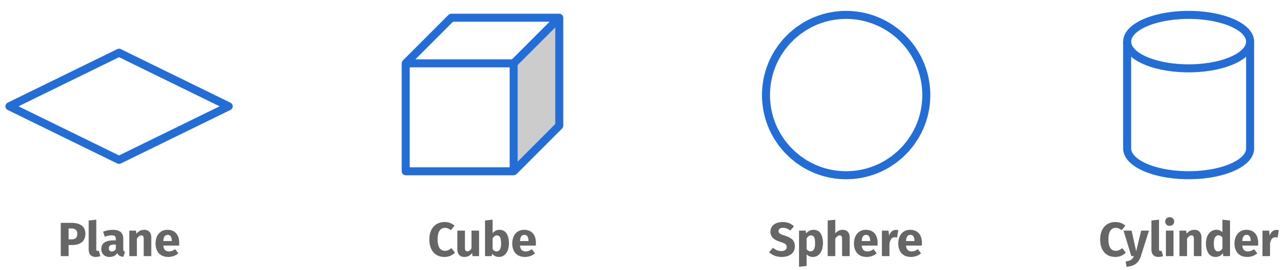


Explicit UV Mapping

- Generally requires an artist to manually create the mapping between the mesh and the texture

Projection Mappings

- We can also use a *Projection Mapping*:
  1. define a mapping between the texture and some intermediate surface:

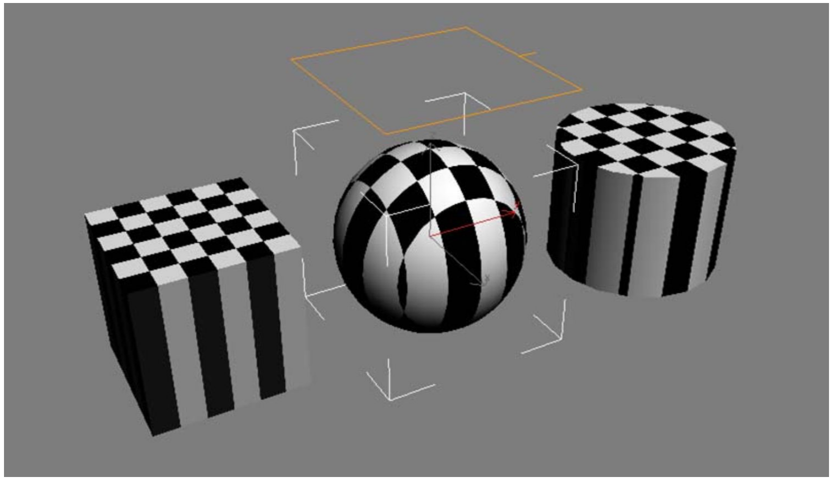


  2. Project intermediate surface onto 3D object surface

Planar Mapping

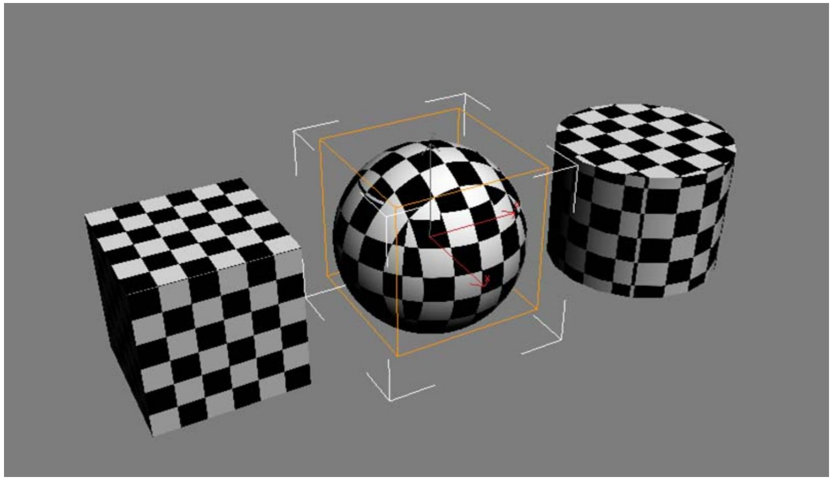
1. We can project along the xz plane by ignoring the y coordinate and taking only (x, z)
2. To project along a plane of size $w \times h$:

$$(u, v) = (x/w, z/h)$$

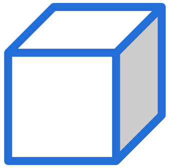


Triplanar / Box Mapping

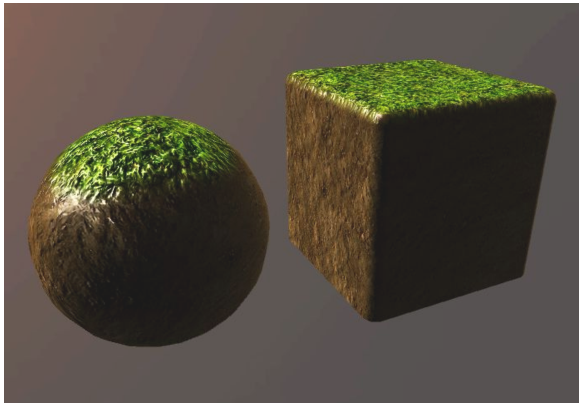
- Project 3 planes, according to the orientation of the surface (using surface normals)
- Texture map is never projected from an angle of more than 45° from the normal - no “smearing”



Triplanar / Box Mapping

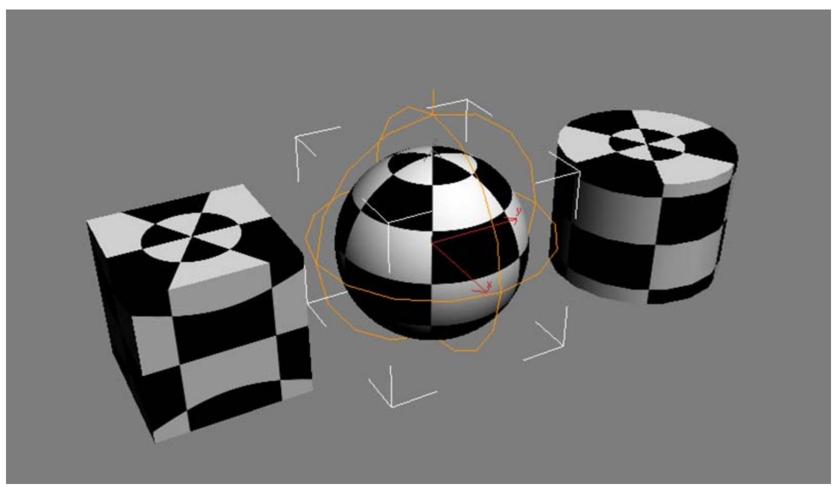


- Triplanar mapping is often used for terrain
- For example, we can project a grass texture along the xz plane and a rock texture along the other 2 axes:



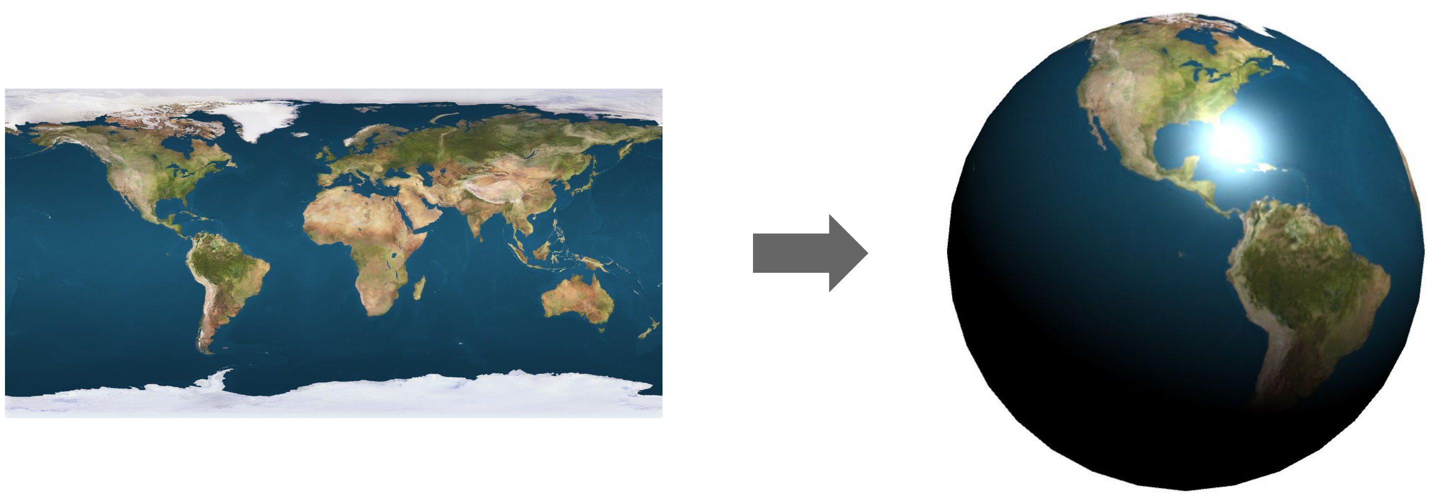
Spherical Mapping

1. Map 3D cartesian coordinates to spherical coordinates
2. Ignoring the radius, we get 2D coordinates which we can then use to sample the texture

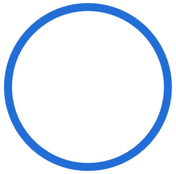


Spherical Mapping

- Useful for spherical objects (obviously!)
- As long at the object remains centered on its origin, the texturing stays the same independent of scale

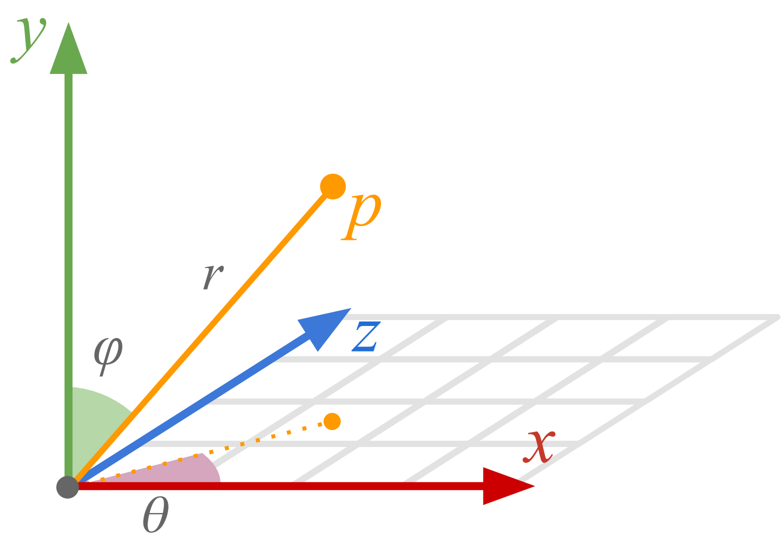


Spherical Mapping

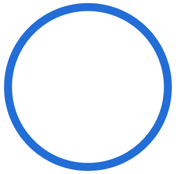


1. Given cartesian coordinates (x, y, z) of point p we want to find its spherical coordinates (r, θ, φ):

$$r = \sqrt{x^2 + y^2 + z^2} \qquad \theta = \mathrm{atan2}(z, x) \qquad \varphi = \mathrm{acos}(y / r)$$

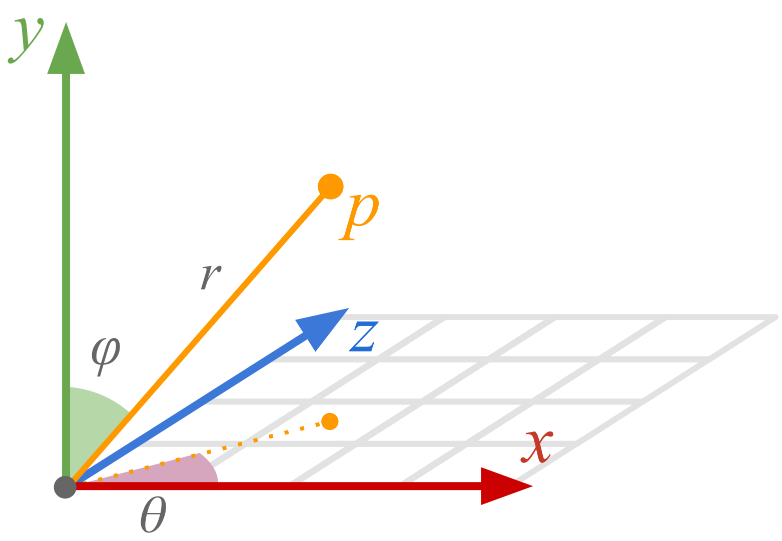


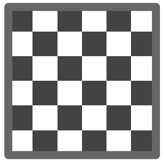
Spherical Mapping



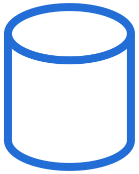
2. To project through the unit sphere, we can ignore r and normalize to $[0,1]$:

$$u = 0.5 + \theta / 2\pi \qquad v = 1 - \varphi / \pi$$

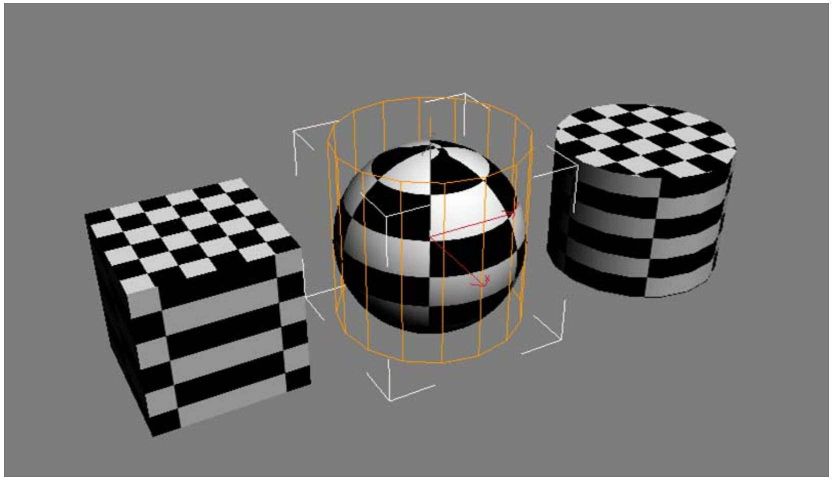




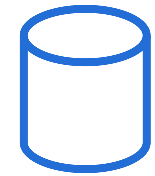
Cylindrical Mapping



1. Map 3D cartesian coordinates to cylindrical coordinates
2. Ignoring the radius, we get 2D coordinates which we can then use to sample the texture



Cylindrical Mapping



1. To find cylindrical coordinates (r, θ, z'):

$$r = \sqrt{x^2 + z^2} \qquad \theta = \mathrm{atan2}(z, x) \qquad z' = y$$

2. To project through a unit cylinder of height h, we can ignore r and normalize to $[0,1]$:

$$u = 0.5 + \theta \,/\, 2\pi \qquad v = z' \,/\, h$$

✱ Cylinder caps can be filled in with plane mapping

Texture Sampling

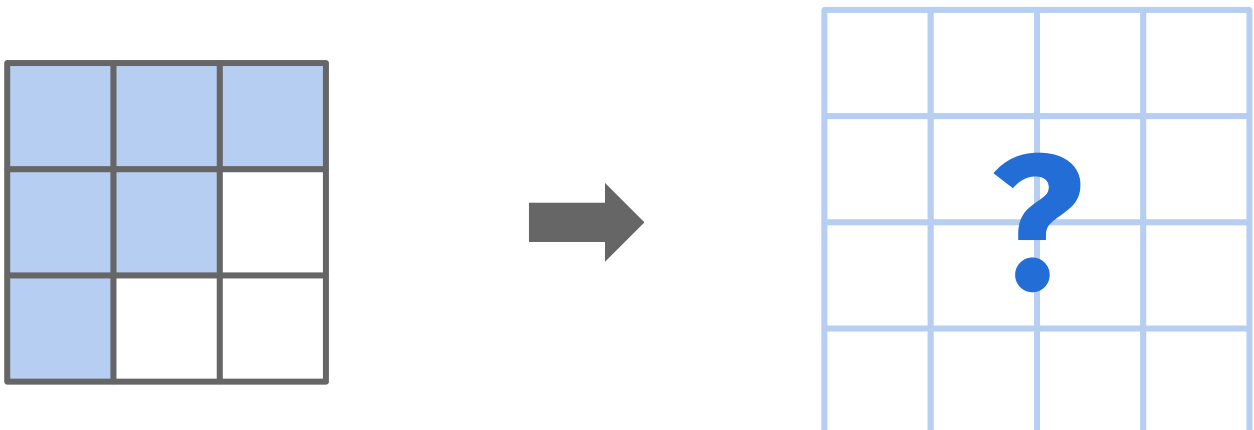
Computer Graphics 2020

Sampling Textures

- Assuming we have our mapping f from a 3D location to a 2D point in texture space, we now need to sample the texture at this point
- If our coordinates in texture space fall on a specific Texel - we just return its value
- Usually that will not be the case - we will need to either magnify or minify the texture!

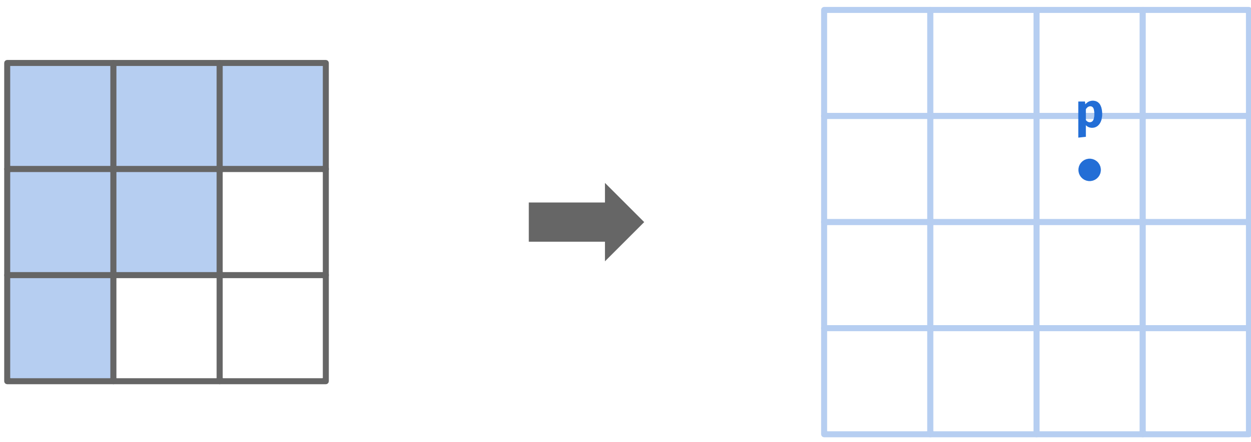
Magnification

- Assume we have a texture of size 3×3 texels and we want to render it to a 4×4 screen pixel area:



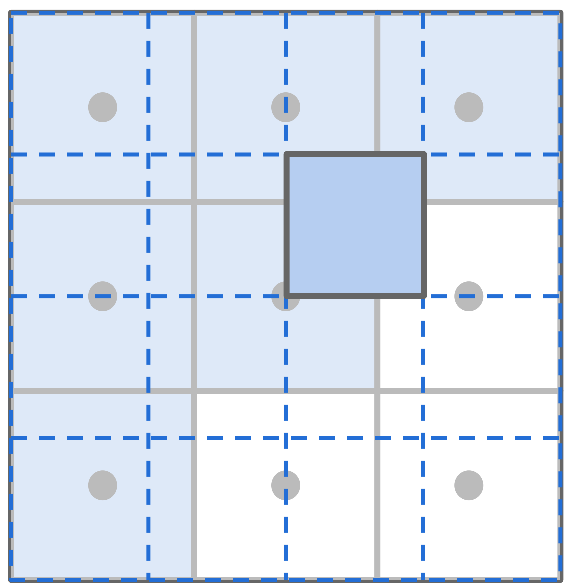
Magnification

- Consider the pixel p. What color should it be?



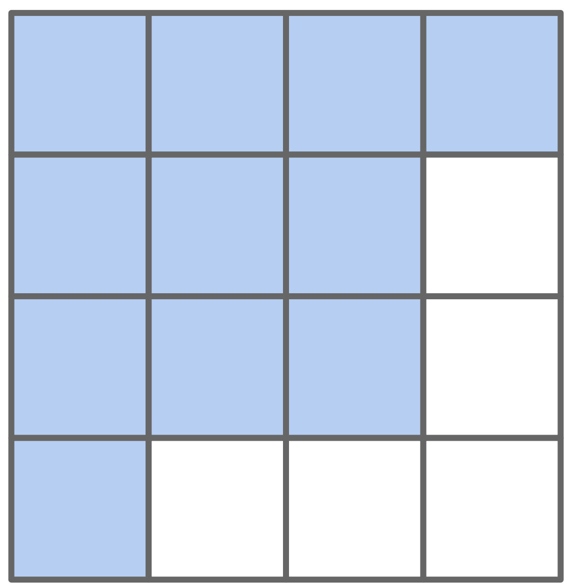
Magnification - Nearest Neighbor

- The simplest idea - find the closest texel and take its value. This is called ***nearest neighbour interpolation***



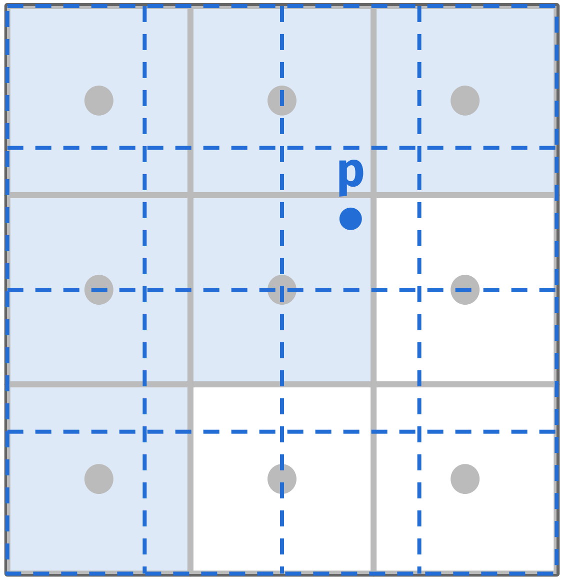
Magnification - Nearest Neighbor

- This is method is really fast and simple, but we get jagged edges and artefacts



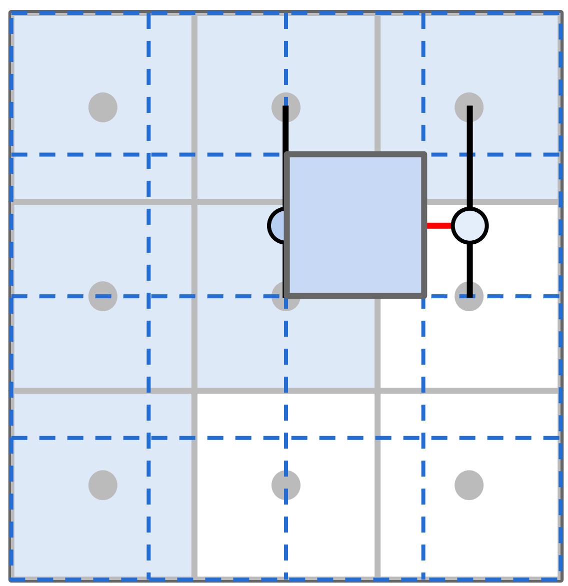
Magnification - Bilinear

- A better idea - find the 4 closest texels and interpolate between them



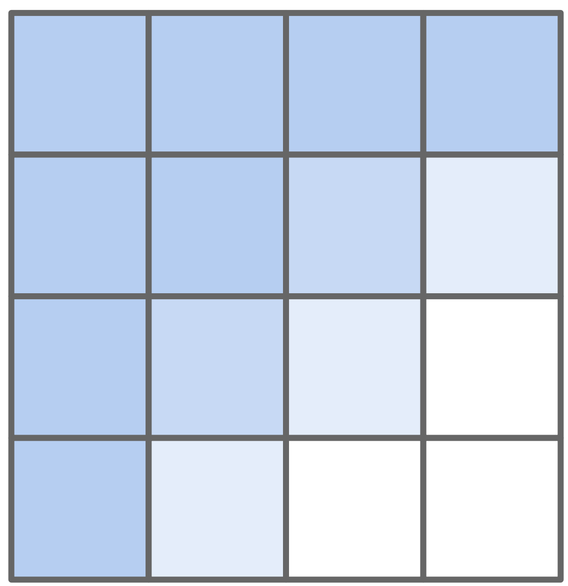
Magnification - Bilinear

- First interpolate linearly in one direction, then interpolate the results in the other direction. This is called ***bilinear interpolation***



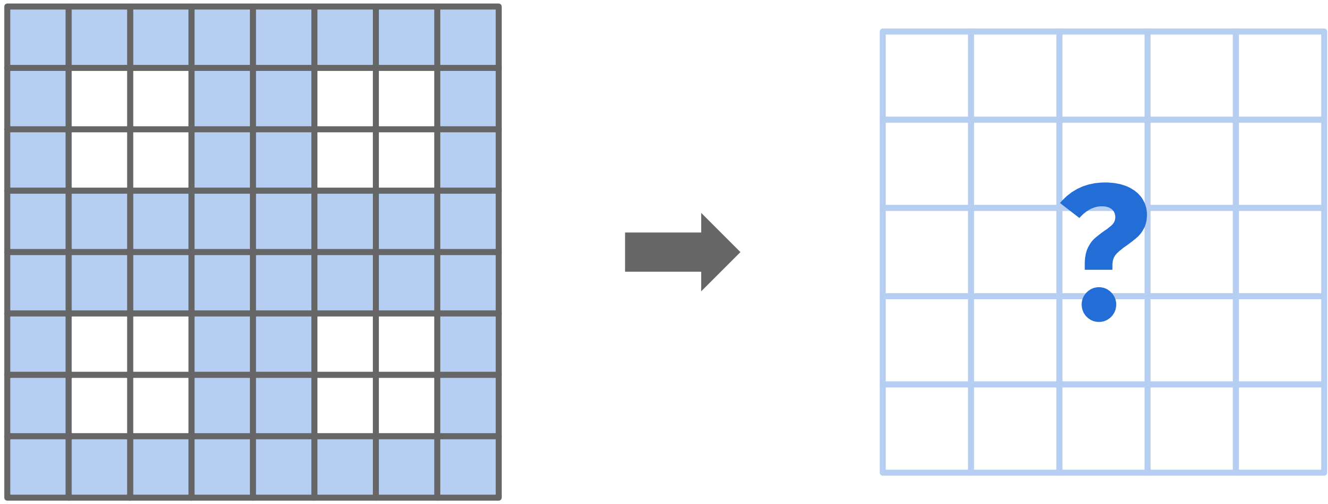
Magnification - Bilinear

- This method gives much smoother results
- No jagged artefacts



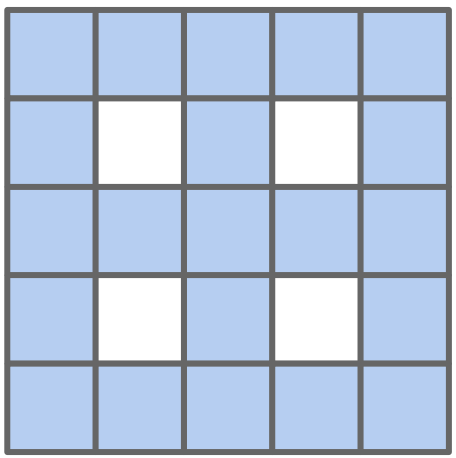
Minification

- Now we have a texture of size 8×8 texels and we want to render it to a 5×5 screen pixel area:

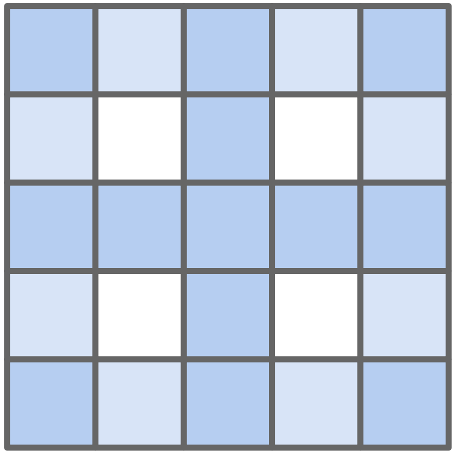


Minification

- We can use the same simple approaches, which work quite well in this case:



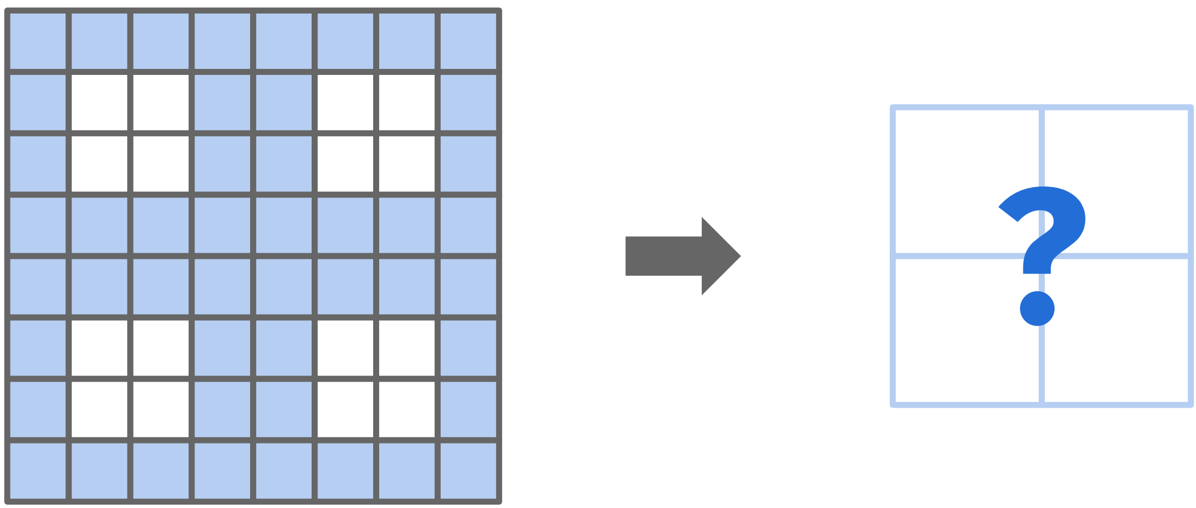
Nearest Neighbor



Bilinear Interpolation

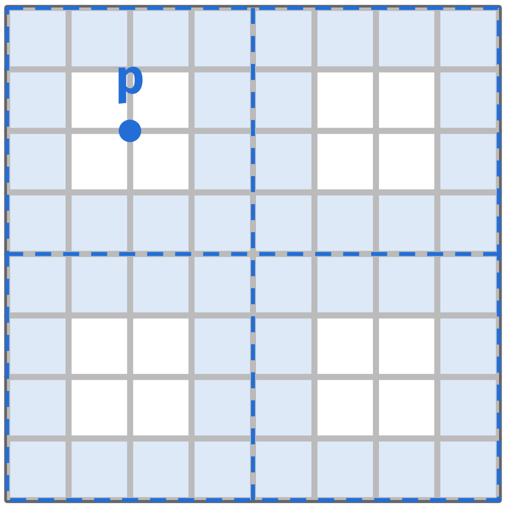
Minification

- But what if we want to render the texture to a smaller area, say 2×2 screen pixels?



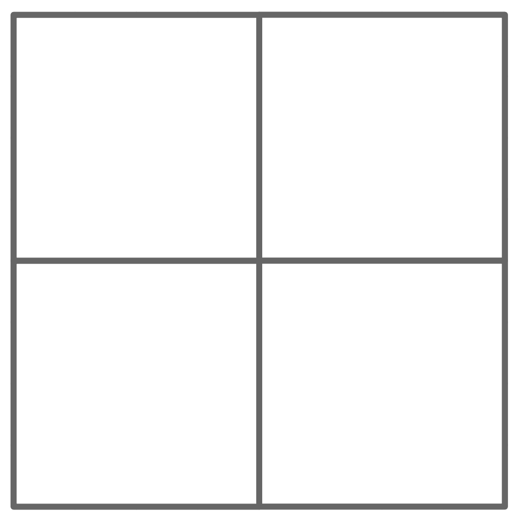
Minification

- When upscaling, we always have 4 neighbours
- When downscaling by a factor of 2 or more, we have more than 4 neighbours that affect the final value!



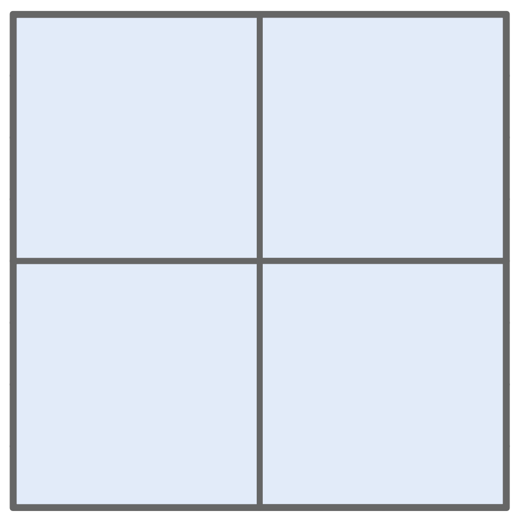
Minification

- If we use regular bilinear interpolation, we can get bad results (completely white image!)
- the effect can be even worse when using nearest neighbor



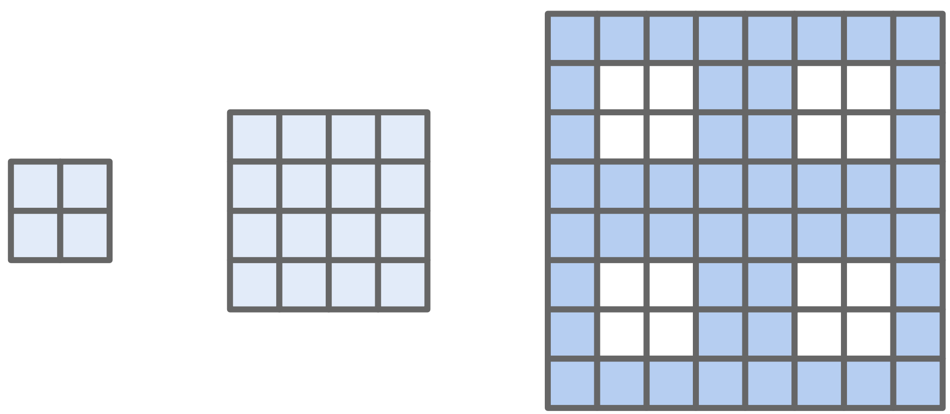
Minification

- To get better downsampling, we can interpolate between all the texels that affect the final value
- This can be very inefficient, especially for real time graphics



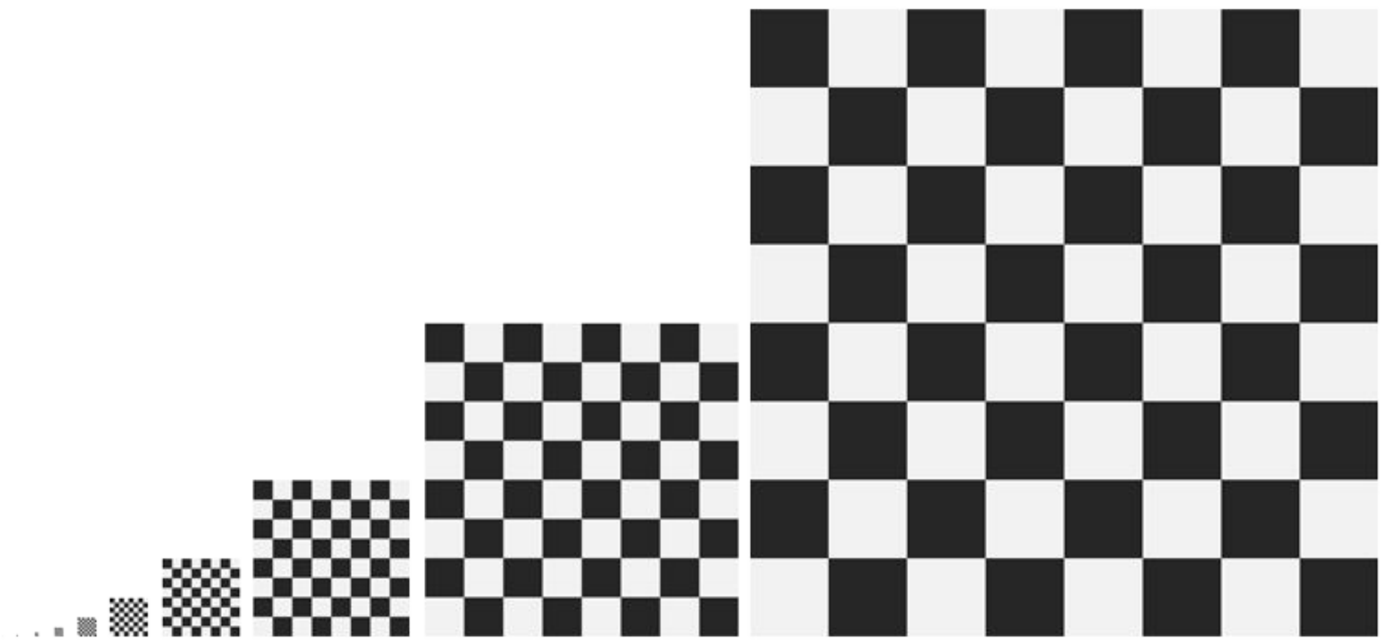
Mipmaps

- To solve this, we pre-compute ***Mipmaps***, which store the same texture at different resolutions (powers of 2 usually)
- We can then look up the appropriate image based on the projected size

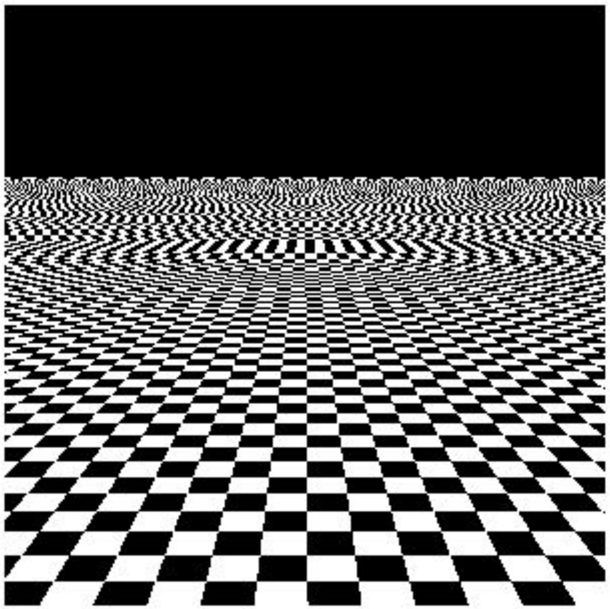


Mipmaps

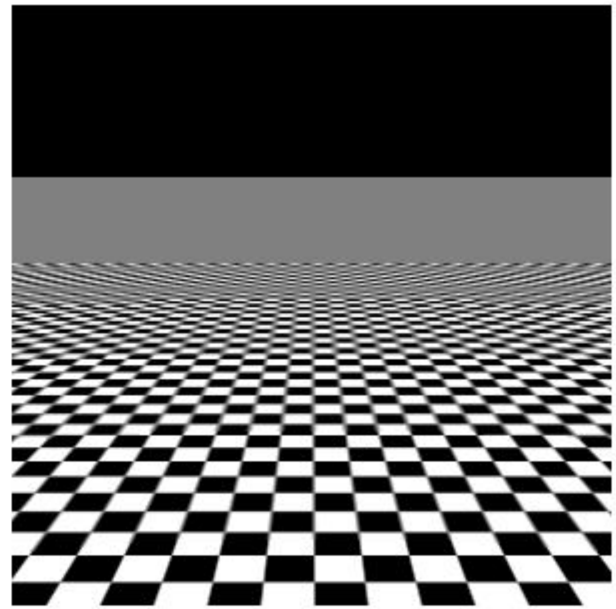
- The name “Mipmap” comes from the Latin phrase *Multum In Parvo* meaning “much in little”



Mipmaps



Point sampling



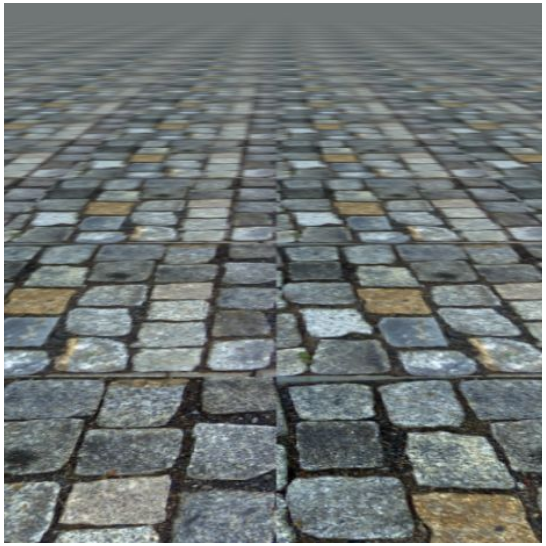
Bilinear interpolation + Mipmaps

Anisotropic Filtering

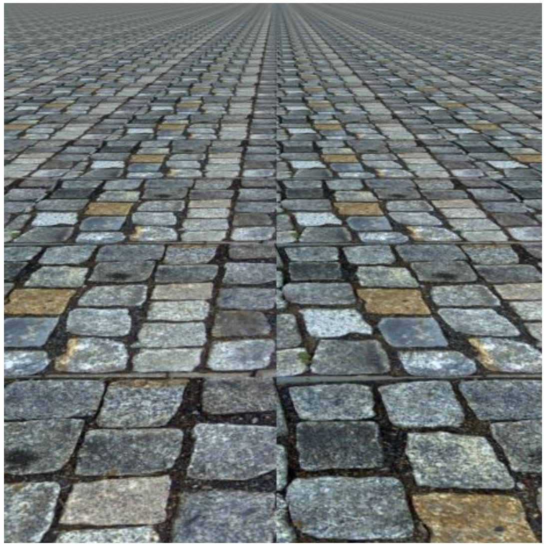
- When using ***Anisotropic Filtering***, in addition to downsampling the texture evenly, the texture is downsampled on each axis independently
- Useful for extreme viewing angles

Anisotropic Filtering

- Enhances the image quality of textures at oblique viewing angles



Regular Mipmap



Anisotropic filtering