Textures

Computer Graphics
Instructor: Sungkil Lee

Today

- **Textures**
  - Why we need textures?
  - A simple example
- **Texture mapping**
  - Diffuse texture mapping
  - Forward vs. backward texture mapping
- **Texture sampling and filtering**
  - Aliasing and antialiasing

Textures

Textures?

- **A “texture” is a repeated pattern in some spaces.**
  - In physical space, the texture indicates repeated ***image*** patterns.
  - In temporal space, it refers to repeated sound patterns.

- **Examples**
  - Wall papers with repeated patterns
  - "Video textures", SIGGRAPH 2000

- **In CG, textures usually refer to images in GPU memory.**
  - Initially, they meant repeated spatial patterns in the geometric surfaces
  - Nowadays, they simply mean images in GPU memory.
    - e.g., texture memory dedicated to GPU

Why We Need Textures?

- **The materials of objects may vary across the surface.**
  - So, we need to make shading parameters vary across the surface.
  - Geometric modeling can be too costly to represent such materials.
  - We need a more efficient/simpler approach for visually complex materials.
    - We will decouple the complexity of geometries and materials.

Why We Need Textures?

- **Limits of geometric modeling**
  - Geometric modeling can be too costly to represent visually complex objects, including varying materials.
    - Examples: Clouds, grass, terrain, skin
  - We need a more efficient and simpler way for visually complex materials.

Simple Example

- **Modeling of an orange**
  - Consider the problem of modeling an orange
  - Start with an orange-colored sphere: too simple
  - Does not capture surface characteristics (small dimples)

Simple Example

- **Replace sphere with a more complex shape**
  - Takes too many polygons to model all the dimples
- **Take a picture of a real orange, scan it, and "paste" onto simple geometric model**
  - This process is known as texture mapping

Simple Example

- **Still might not be sufficient, because resulting surface will be smooth**
  - Need to change local shape: bump mapping or displacement mapping

Texture Mapping

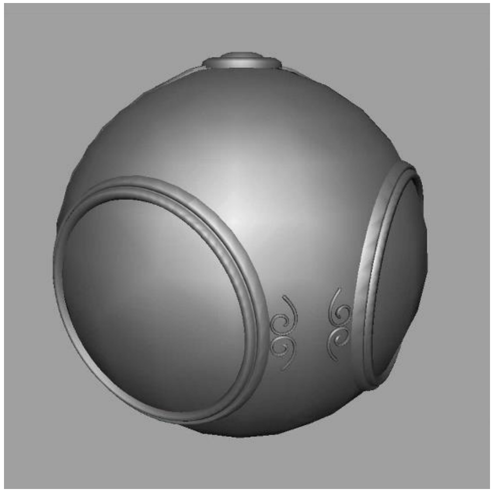
Definition

- **Definition of texture mapping:**
  - a technique of defining surface materials (especially shading parameters) in such a way that vary as a function of position on the surface.

- **Very simple in comparison to geometric modeling.**
  - However, it produces complex-looking effects at reduced cost.

Diffuse Texture Mapping

- **Diffuse mapping**
  - Uses image colors to fill inside of polygon.
  - This common texture mapping usually refers to the diffuse mapping.



geometric model

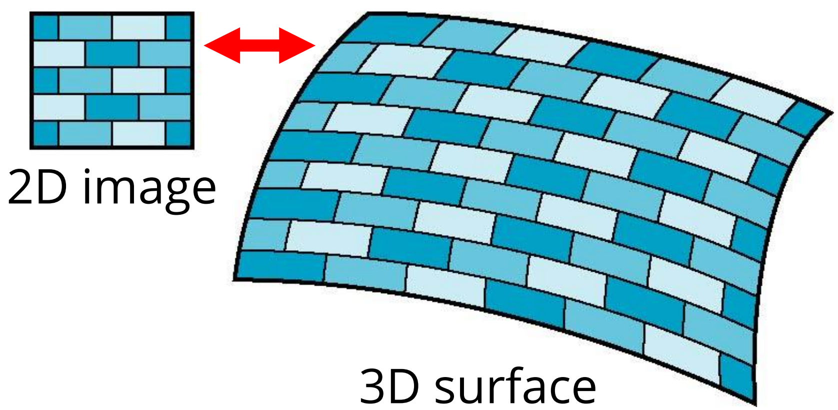
diffuse-texture mapped

- **Example: wood gym floor with smooth finish**
  - Color varies with surface positions (use an image)

Mapping Textures to Surfaces

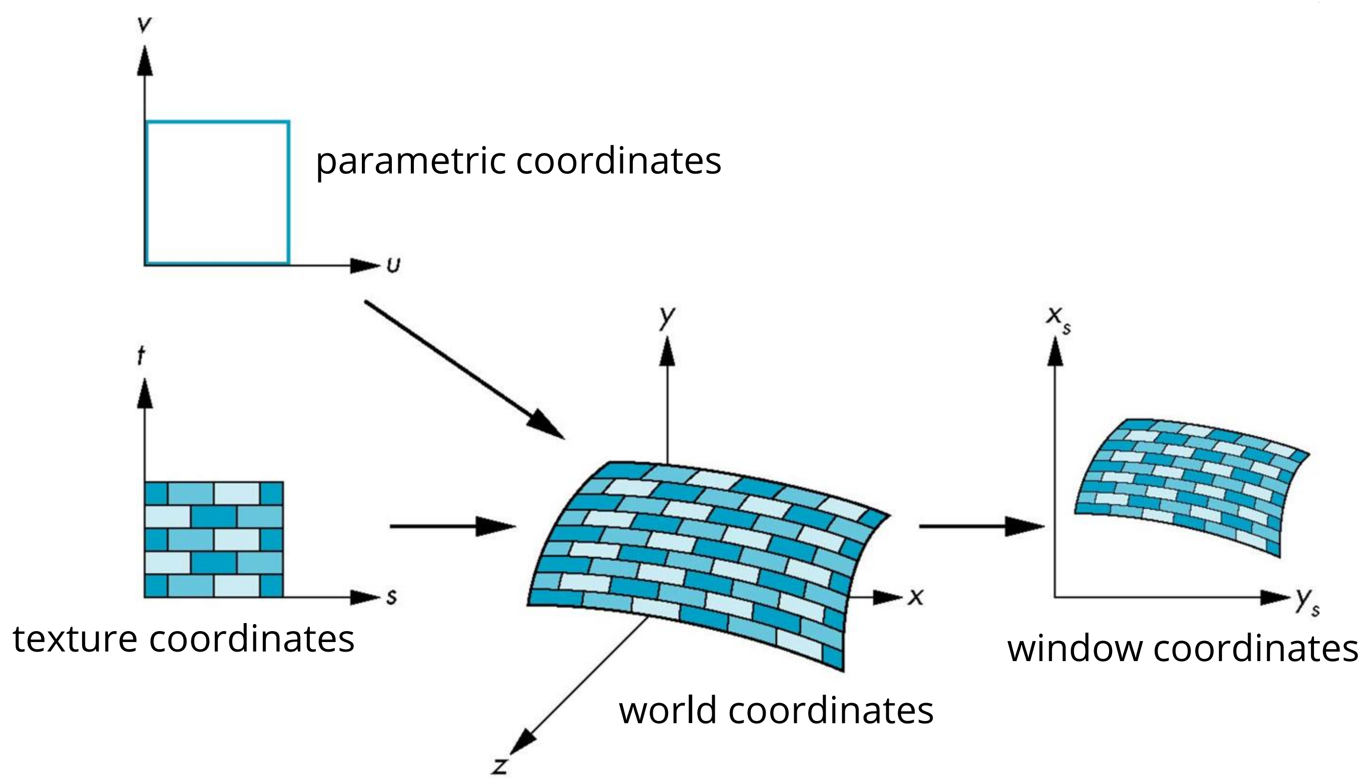
- **Texture mapping: 2D image to 3D surface**
  - Texture: a 2D function of (s, t) or (u, v)
  - 3D Surface: a 3D function of (x, y, z)

- **We need to map the texture to the surface.**
  - This involves forward or backward mapping approaches.



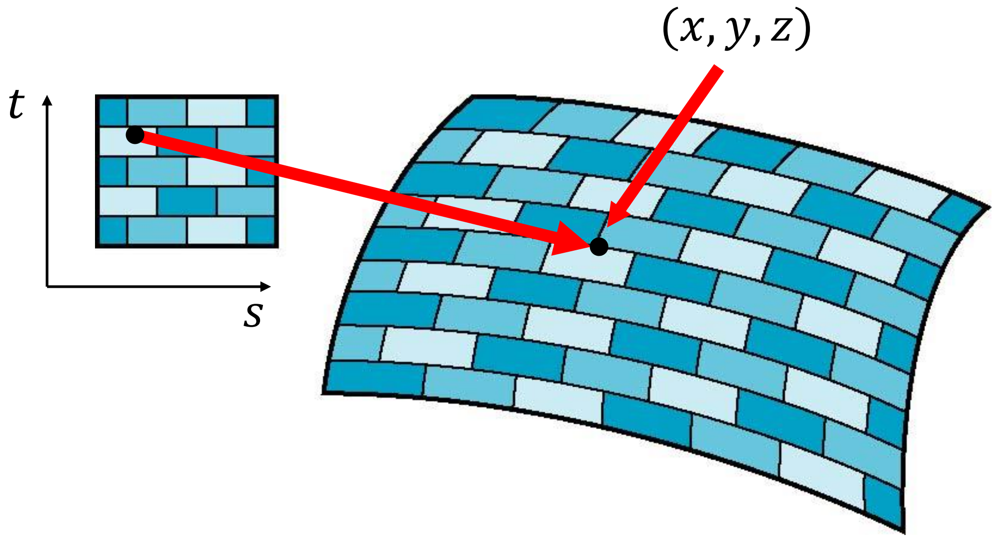
Coordinate Systems

- **Coordinate systems for texture mapping**
  - Parametric coordinates: to model curves and surfaces
  - Texture coordinates: identify points in the image
  - World Coordinates: conceptually, where the mapping takes place
  - Window Coordinates: where the final image is really produced



Forward Mapping

- **Forward mapping involves a projection from the image to the surface.**
  - Appear to need three functions: $x = x(s,t)$, $y = y(s,t)$, $z = z(s,t)$



  - But, we want to go the other way, because we need to look up texture coordinates from surface positions during the shading.

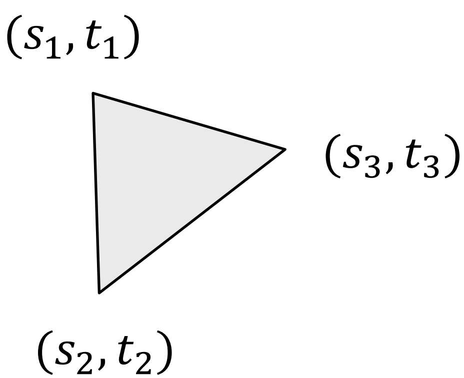
Backward Mapping

- **Backward mapping:**
  - Given a point on an object, we want to know to which point in the texture it corresponds. We need a map of the form:

$$s = s(x,y,z) \quad t = t(x,y,z)$$

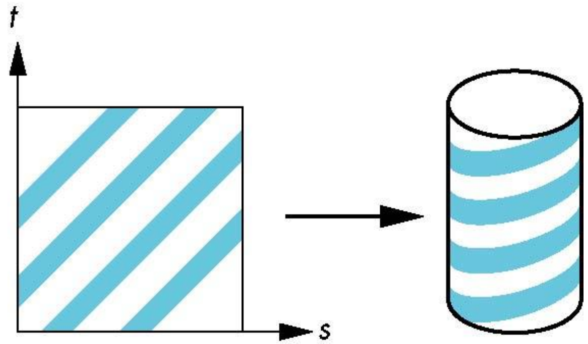
  - Such functions are difficult to find in general. For a complex shape, we use pre-defined texture coordinate mapping.

- **Vertex attributes include (s,t) for texture mapping.**
  - Texture coordinates are interpolated via rasterizer during the rendering.
  - Then, you can fetch the texture using the per-fragment coordinates.



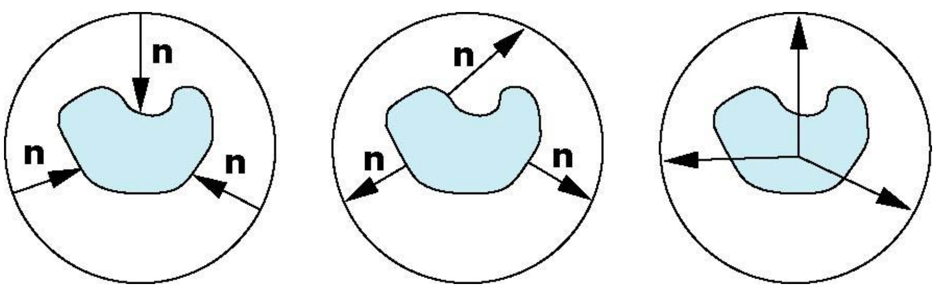
Backward Mapping

- **Backward mapping: a two-step approach**
  - One solution to the mapping problem is to first map the texture to a simple intermediate surfaces, including cylinder and sphere.
    - For example, $x = r\cos 2\pi s, y = r\sin 2\pi\, s, z = h\, t$ for the cylinder.



A mapping from the texture to a cylinder

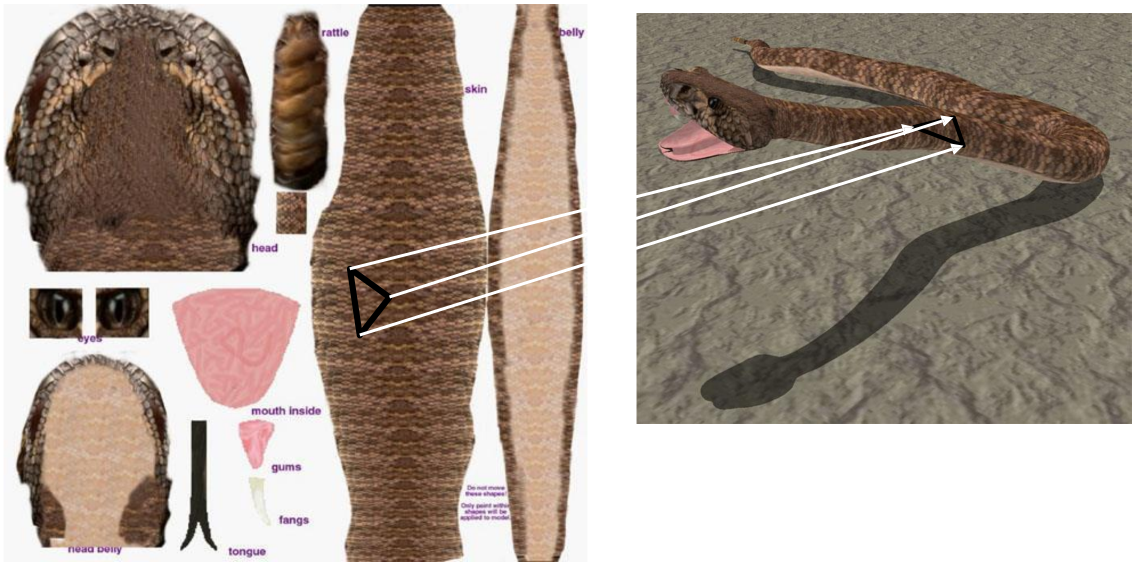
  - Then, we can easily re-project the intermediate surfaces to the object.



How to Do in Practice

- **Practically, we do not handle the texture mapping directly.**
  - Use pre-defined texture coordinates, relying on the 3D modeling artists.
  - The majority of 3D models are provided with texture coordinates.
  - Hence, just use the predefined texture coordinates as they are.

- **Example of texture atlas, common in games.**



Texture Sampling and Filtering

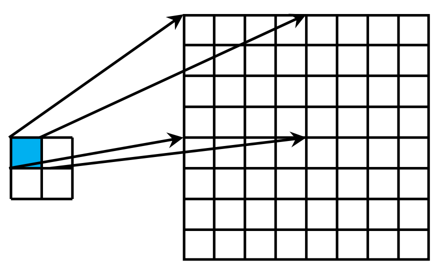
Resampling

- **Both *textures* and *fragments* of surfaces are raster images.**
  - Recall raster images are sampled representation of continuous function.

- **Aliasing:**
  - Insufficient sampling rates may cause the incorrect reconstruction of continuous signals.
  - Hence, we need resampling, when the resolutions of textures and fragments do not match.

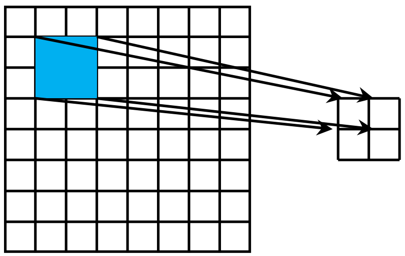
Magnification vs. Minification

- **Texture magnification:**
  - Texture resolution < object surface resolution
  - In this case, aliasing is not that severe, but we may get better reconstruction by interpolation.
- **Texture minification:**
  - Texture resolution > object surface resolution
  - Aliasing is a serious problem; we need to pre-integrate the signals for better reconstruction.



Texture Polygon

Magnification



Texture Polygon

Minification

Magnification

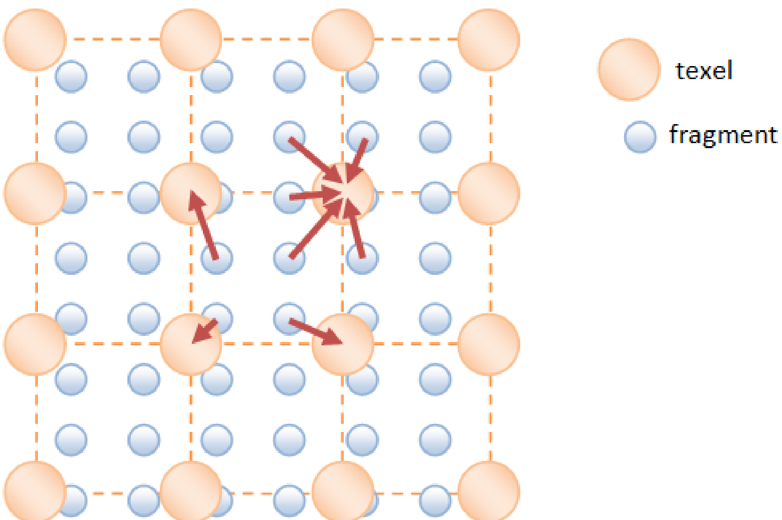
- **Adjacent pixels in the window space map to the same texel.**
  - We can use the texel as it is: nearest neighbor sampling
  - We can apply interpolation according to the window space position.
    - Bilinear, bicubic interpolation, …

[Akenine-Moeller et al.: Real-time Rendering]

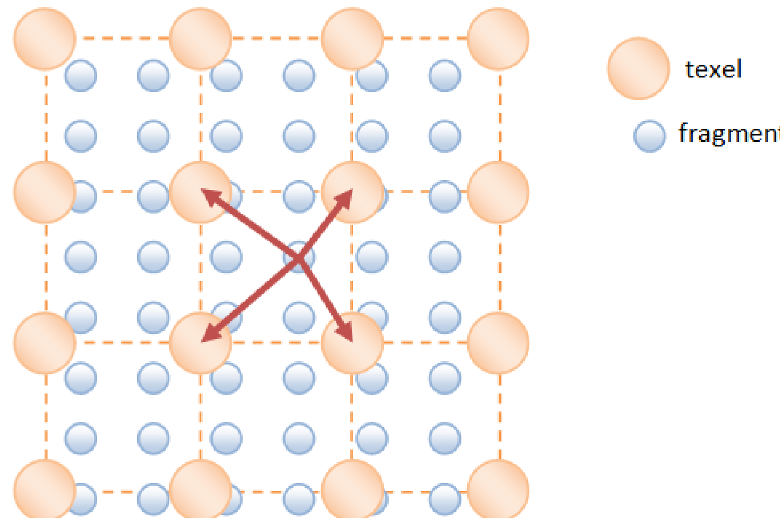
nearest neighbor	bilinear interpolation	bicubic interpolation

Magnification

- **Point (nearest-neighbor) sampling vs. Bilinear interpolation**



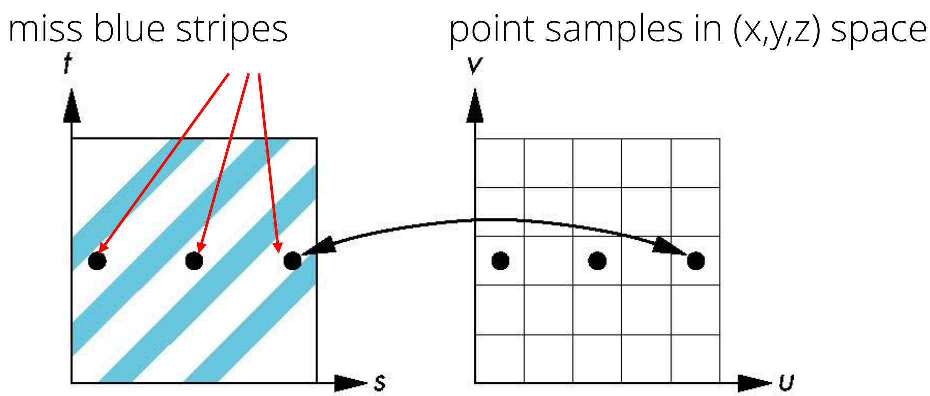
Magnification – Nearest Point Sampling

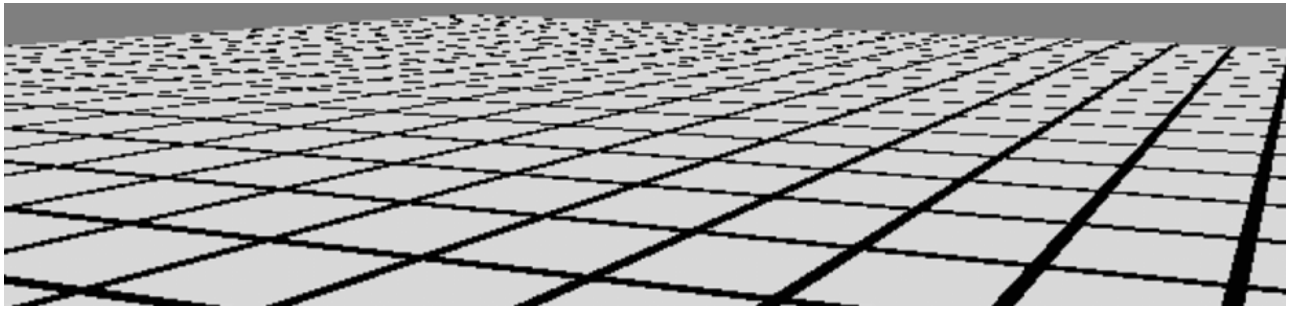


Magnification – Bilinear Interpolation

Minification

- **The aliasing problem from undersampling**
  - Many samples correspond to the single fragment, but point sampling only fetches a single sample among them.

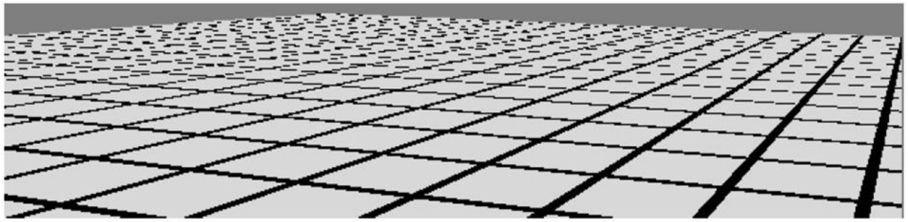




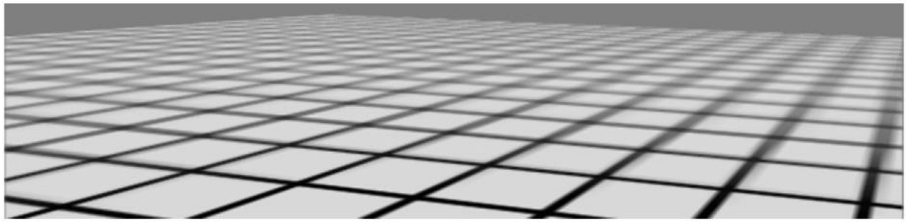
Undersampling in minification is the most pronounced at the farther surfaces, which suffers from aliasing.

Pre-integration for Minification

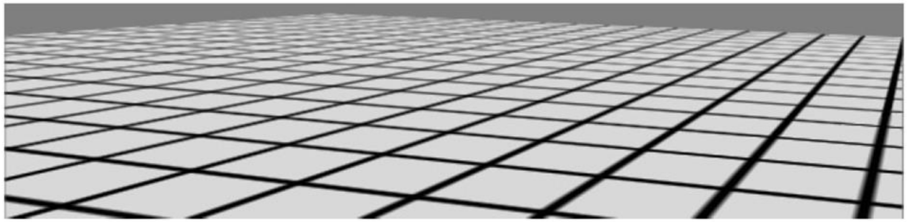
- **Pre-integration (area averaging)**
  - Prior to the texture look-up, we can average the multiple texels.
  - **Mipmapping** is a a standard pre-integration technique in OpenGL



nearest neighbor (point sampling)



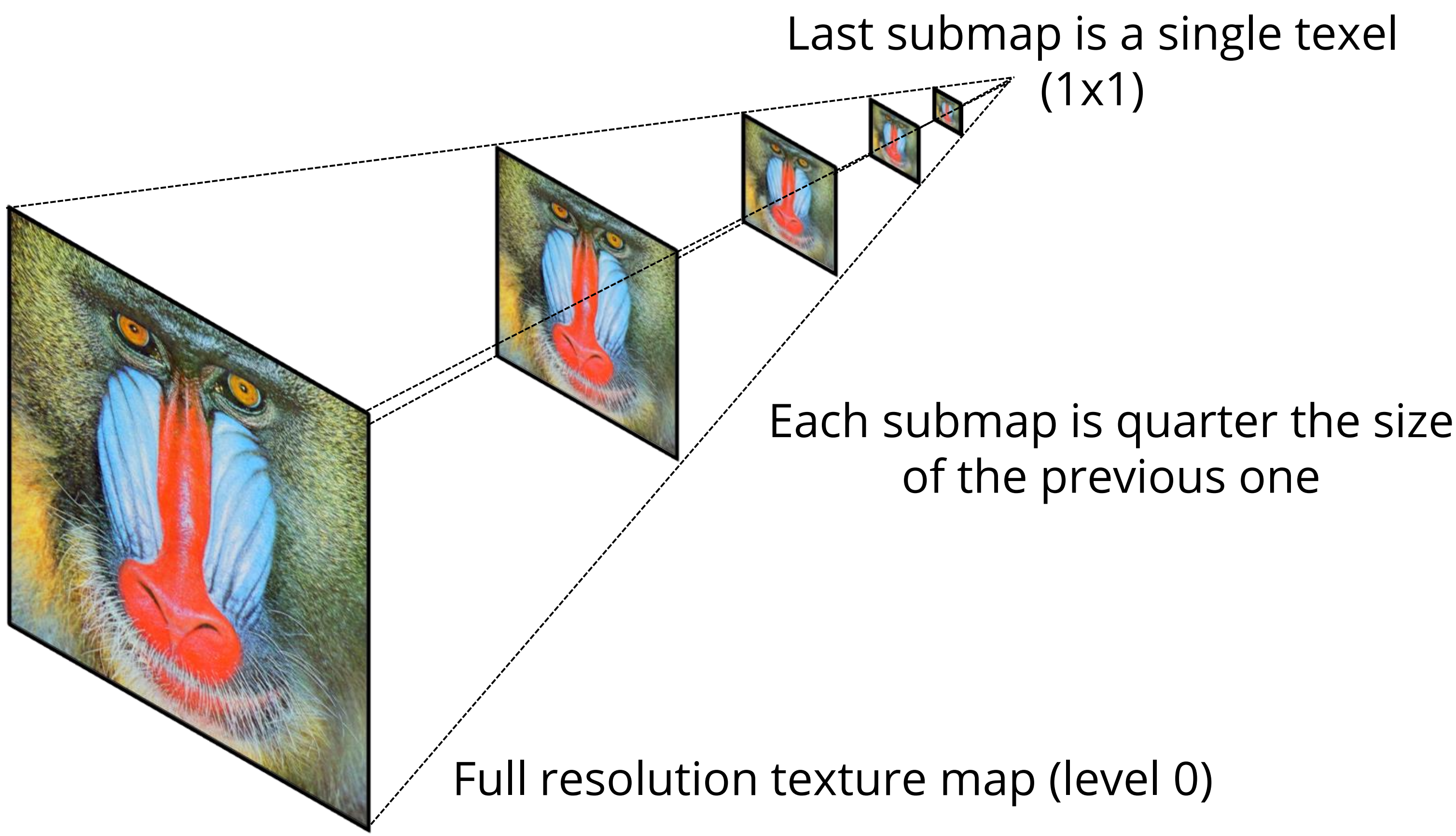
mipmapping



summed-area tables (precise averaging)

Pre-integration via Mipmapping

- **A texel in a mipmap level n represents spatial averages of 2^n.**
  - Thereby, we can fetch a pre-integrated single texel in a area.
  - To find a mipmap level, we need a bit of computation, but this is automatically provided in OpenGL.



Tri-linear Interpolation in Mipmapping

- **Trilinear interpolation: looking up non-integer mipmap level**
  - A mipmap level can be a real number, but mipmaps are pre-built for integer levels.
  - OpenGL (with **GL_LINEAR_MIPMAP_LINEAR**) linearly interpolates between two (bilinearly filtered) integer levels. For example, level 1.5 interpolates levels 1 and 2, by the factor of 0.5.