Texture

- Instead of having each surface as a solid colour, we can attach "texture" to the surface.
- Texture mapping refers to putting an image onto a surface.
- e.g. putting brick or wood grain patterns onto a surface
- There are also other application as well.

Digital Images

- Textures are often specified by digital images.
- A digital image is a rectangular array ($N \times M$) of pixel values.
- For grayscale (also called monochromatic or luminance) images, each pixel is a value in $[0, 255]$ (8-bit images). Black is 0, white is 255.
- For colour images, each pixel is typically specified by a vector of (R, G, B) values in $[0, 255]$ (24-bit images).
- Images may be stored in different formats (e.g. GIF, TIFF, PNG, PDF, JPEG).
- Most formats perform some data compression, some are lossy (e.g. JPEG).
- OpenGL: need external libraries to load image data.

Texture Mapping

- For realistic rendering of objects, texture needs to be applied to surfaces.
- Texture mapping: uses an image to influence the colour of a fragment.
- Done inside the fragment shader.

Texture Mapping

- Textures are patterns that may repeat periodically.
- Textures can be specified in one, two, three, or four dimensions.
- We will only look at 2D texture mapping.

2D Texture Mapping

- Texture is a 2D image (can be loaded or generated by code).
- It is an array of texture elements (texels).
- The texture can be thought of as an array $T(s,t)$ where s and t are texture coordinates. We assume all texture coordinates are real numbers in $[0,1]$.
- A texture map is a set of functions that map coordinates on the surface to texture coordinates:

$$(s,t) = (s(x,y,z,w), t(x,y,z,w))$$

- Conceptually: for each fragment on the object, the texture map tells us which texel should be used.

2D Texture Mapping

- One particular issue: it may be that (s, t) is "in between" texels. Which texel should we use?
- Simplest: point sampling (`GL_NEAREST`)—use the nearest one. Can lead to visible aliasing effects.
- More complicated: linear filtering (`GL_LINEAR`)—interpolates between close texels.

2D Texture Mapping

- If the texture coordinates are outside of $[0, 1]$, we can
  - `GL_REPEAT`: repeat the pattern periodically
  - `GL_MIRRORED_REPEAT`: repeat the pattern periodically but reflect each time
  - `GL_CLAMP_TO_EDGE`: coordinates clamped to 0 and 1.
  - `GL_CLAMP_TO_BORDER`: coordinates outside are given a "border" colour.
- These can be specified independently for s and t.
- Specified with `glTexParameteri`.

Mipmapping

- Depending on the distance of the object to the viewer, the size of a fragment may be much larger or smaller than a texel.
- If a texel is larger than one pixel, minification is needed.
- If a texel is smaller than one pixel, magnification is needed.
- This can be controlled using `GL_TEXTURE_MIN_FILTER` and `GL_TEXTURE_MAG_FILTER` using point sampling or linear filtering, but there is a different way.
- Mipmapping: generates a set of texture arrays from the original, at different resolutions (`glGenerateMipmap`).
- Use `GL_LINEAR_MIPMAP_LINEAR` for minification to automatically use the "right size". Second parameter is to interpolate amongst the different resolutions.

OpenGL Texture Mapping Setup

- Call `glGenTextures` and `glBindTexture`.
- Set up wrapping, minification and magnification parameters.
- Load image and set texture with `glTexImage2D`.
- Call `glGenerateMipmap` to generate mipmaps.
- Provide a 2 dimensional input atrribute in the vertex shader for the texture coordinates.
- Load the texture coordinates of each vertex using a VertexAttribArray.

OpenGL Texture Mapping Setup

- In vertex shader, accepts texture coordinates and perform needed calculations (usually none) and pass the texture coordinates to fragment shader (interpolated for each fragment).
- In fragment shader, there is automatically a uniform `sampler2D` parameter.
- In fragment shader, the function `texture(uTextureMap, vTexCoord)` will return the texel value using the appropriate sampling selected.
- The colour returned can be used to determine the colour of the fragment (possibly together with lighting information).

3D Texture Mapping

- Sometimes attaching 2D textures to each surface can look unrealistic (e.g. edge between surfaces)
- Instead, we can define 3D textures for entire object (e.g. wood grain, stone, etc.).
- There will be three texture coordinates (s, t, r).
- The effect will be similar to "carving" an object out of a textured material.

Environment/Reflection Map

- How do we render a scene when there is a highly reflective surface (e.g. mirror)?
- We cannot render the mirror without knowing the rest of the scene.
- Texture mapping can be used to make a good approximation.

Environment/Reflection Map

- Looking at a mirror: the angle of reflection is known given the viewer position.
- We can first pretend to render the scene without the mirror.
- Place the camera at the centre of the mirror, looking towards the normal vector of the mirror.
- Render the scene. This is what the mirror "sees".
- Use the scene as a texture to map onto the surface of the camera.
- Problems:
  - rendering first pass without the mirror may be unrealistic
  - what should be the projection plane in the first rendering?
- More advanced techniques are needed to solve these problems.

Bump Mapping

- Sometimes we want a surface to appear non-smooth with little "bumps". e.g. the peel of an orange is not flat.
- The small variations are hard to model geometrically.
- However, lighting variation can be used to "fool" the viewer: if the normal vectors are perturbed, it will appear that surface is "bumpy".
- This is applied in the fragment shader.

Bump Mapping

- We will omit the mathematics to compute.
- The perturbations to the normal vectors can be stored as a texture called a normal map.
- The normal map is used to modify the normal vectors before lighting calculations.