Many uses of texture mapping
Define variation in surface reflectance

Pattern on ball
Wood grain on floor
Describe surface material properties

Multiple layers of texture maps for color, logos, scratches, etc.
Normal mapping

Use texture value to perturb surface normal to give appearance of a bumpy surface.
Observe: smooth silhouette and smooth shadow boundary indicates surface geometry is not bumpy.

Rendering using high-resolution surface geometry
(note bumpy silhouette and shadow boundary)
Represent precomputed lighting and shadows

Grace Cathedral environment map

Environment map used in rendering
Texture coordinates

“Texture coordinates” define a mapping from surface coordinates (points on triangle) to points in texture domain.

\[ \text{myTex}(u, v) \]

is a function defined on the \([0,1]^2\) domain:

\[ \text{myTex} : [0, 1]^2 \rightarrow \text{float3} \]

(represented by 2048x2048 image)

Eight triangles (one face of cube) with surface parameterization provided as per-vertex texture coordinates.

Final rendered result (entire cube shown).

Location of triangle after projection onto screen shown in red.

Location of highlighted triangle in texture space shown in red.

Today we’ll assume surface-to-texture space mapping is provided as per vertex values (Methods for generating surface texture parameterizations will be discussed in a later lecture)
Visualization of texture coordinates

Texture coordinates linearly interpolated over triangle

(0.0, 1.0) (green)
(0.0, 0.0)
(1.0, 0.0) (red)
More complex mapping

Visualization of texture coordinates

Triangle vertices in texture space

Each vertex has a coordinate \((u,v)\) in texture space.
(Actually coming up with these coordinates is another story!)
Simple texture mapping operation

for each covered screen sample \((x,y)\):
  
  \[(u,v) = \text{evaluate texcoord value at } (x,y)\]

  \[
  \text{float3 texcolor} = \text{texture.sample}(u,v); \]

  set sample’s color to texcolor;
Texture mapping adds detail

Rendered result

Triangle vertices in texture space
Texture mapping adds detail

rendering without texture  rendering with texture  texture image

Each triangle “copies” a piece of the image back to the surface.
Another example: Sponza

Notice texture coordinates repeat over surface.
Textured Sponza
Example textures used in Sponza
Texture space samples

Sample positions in XY screen space

Sample positions are uniformly distributed in screen space (rasterizer samples triangle's appearance at these locations)

Sample positions in texture space

Texture sample positions in texture space (texture function is sampled at these locations)
Recall: aliasing

Undersampling a high-frequency signal can result in aliasing.

1D example:

\[ f(x) \]

2D examples:
Moiré patterns, jaggies
Aliasing due to undersampling texture

No pre-filtering of texture data (resulting image exhibits aliasing)

Rendering using pre-filtered texture data
Aliasing due to undersampling (zoom)

No pre-filtering of texture data
(resulting image exhibits aliasing)

Rendering using pre-filtered texture data
Filtering textures

- **Minification:**
  - Area of screen pixel maps to large region of texture (filtering required -- averaging)
  - One texel corresponds to far less than a pixel on screen
  - Example: when scene object is very far away

- **Magnification:**
  - Area of screen pixel maps to tiny region of texture (interpolation required)
  - One texel maps to many screen pixels
  - Example: when camera is very close to scene object (need higher resolution texture map)
Filtering textures

Actual texture: 700x700 image (only a crop is shown)

Actual texture: 64x64 image

Texture minification

Texture magnification
Mipmap (L. Williams 83)

Idea: prefilter texture data to remove high frequencies

Texels at higher levels store integral of the texture function over a region of texture space (downsampled images)

Texels at higher levels represent low-pass filtered version of original texture signal
Mipmap (L. Williams 83)

Williams’ original proposed mip-map layout

“Mip hierarchy”
level = d

What is the storage overhead of a mipmap?

Slide credit: Akeley and Hanrahan
Computing $d$

Compute differences between texture coordinate values of neighboring screen samples
Computing $d$

Compute differences between texture coordinate values of neighboring fragments.

\[
\begin{align*}
    du/dx &= u_{10} - u_{00} \\
    dv/dx &= v_{10} - v_{00} \\
    du/dy &= u_{01} - u_{00} \\
    dv/dy &= v_{01} - v_{00}
\end{align*}
\]

\[
L = \max \left( \sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2} \right)
\]

\[
mip\text{-}map\ d = \log_2 L
\]
Sponza (bilinear resampling at level 0)
Sponza (bilinear resampling at level 2)
Sponza (bilinear resampling at level 4)
Visualization of mip-map level
(bilinear filtering only: d clamped to nearest level)
“Tri-linear” filtering

Bilinear resampling:
  four texel reads
  3 lerps (3 mul + 6 add)

Trilinear resampling:
  eight texel reads
  7 lerps (7 mul + 14 add)

Figure credit: Akeley and Hanrahan
Visualization of mip-map level
(trilinear filtering: visualization of continuous d)
Pixel area may not map to isotropic region in texture
Proper filtering requires anisotropic filter footprint

Texture space: viewed from camera with perspective

$L = \max \left( \sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2} \right)$

$mip\text{-}map\ d = \log_2(L)$
Principle of texture thrift

Given a scene consisting of textured 3D surfaces, the amount of texture information minimally required to render an image of the scene is proportional to the resolution of the image and is **independent** of the number of surfaces and the size of the textures.
Summary: texture filtering using the mip map

- **Small storage overhead (33%)**
  - Mipmap is 4/3 the size of original texture image

- **For each isotropically-filtered sampling operation**
  - Constant filtering cost (independent of d)
  - Constant number of texels accessed (independent of d)

- **Combat aliasing with prefiltering, rather than supersampling**
  - Recall: we used supersampling to address aliasing problem when sampling coverage

- **Bilinear/trilinear filtering is isotropic and thus will “overblur” to avoid aliasing**
  - Anisotropic texture filtering provides higher image quality at higher compute and memory bandwidth cost
Summary: a texture sampling operation

1. Compute u and v from screen sample x,y (via evaluation of attribute equations)
2. Compute du/dx, du/dy, dv/dx, dv/dy differentials from screen-adjacent samples.
3. Compute d
4. Convert normalized texture coordinate (u,v) to texture coordinates texel_u, texel_v
5. Compute required texels in window of filter
6. Load required texels (need eight texels for trilinear)
7. Perform tri-linear interpolation according to (texel_u, texel_v, d)

Takeaway: a texture sampling operation is not just an image pixel lookup! It involves a significant amount of math.

All modern GPUs have dedicated fixed-function hardware support for performing texture sampling operations.
Texturing summary

- Texture coordinates: define mapping between points on triangle’s surface (object coordinate space) to points in texture coordinate space

- Texture mapping is a sampling operation and is prone to aliasing
  - Solution: prefilter texture map to eliminate high frequencies in texture signal
  - Mip-map: precompute and store multiple multiple resampled versions of the texture image (each with different amounts of low-pass filtering)
  - During rendering: dynamically select how much low-pass filtering is required based on distance between neighboring screen samples in texture space
    - Goal is to retain as much high-frequency content (detail) in the texture as possible, while avoiding aliasing
What you should know:

- Textures are used for many things, beyond pasting images onto object surfaces.
  - Normal maps (create appearance of bumpy object on smooth surface by giving false normal to the lighting equations)
  - Displacement maps (encode offsets in the geometry of a surface, which is difficult to handle in a standard graphics pipeline)
  - Environment maps (store light information in all directions in a scene)
  - Ambient occlusion map (store exposure of geometry to ambient light for better representation of surface appearance with simple lighting models)
  - Can you think of / discover others?

- Know how to interpolate texture coordinates
- Know how to index into a texture and compute a correct color using bilinear interpolation
- Be able to create a mipmap and store it in memory
- Be able to compute color from multiple levels of mipmaps using trilinear interpolation
- What is the logic behind selecting an appropriate level in a mipmap?
- What can happen if we select a level that is too high resolution? too low resolution?