- Surface Mapping
  - Mipmapping (Lance Williams 83)
    - Problem
      - In texture mapping, what color to use when:
      - One texel maps to many pixels
        - magnification problem
        - artifact: blocky looking surface
      - One pixel maps to many texels
        - minification problem
        - artifact: random “speckly” look
    - Simple solution
      - bilinear interpolation from neighbors
      - better, but not a full solution
    - MIPmapping
      - Multim in parvo: many things in a small place
      - Basic approach:
        - Perform approximate filtering ahead of time
        - During rendering:
          - Figure out the size of the pixel in texture space
          - Use the prefiltered texels to approximate appropriate filter
      - Some details:
        - Prefiltering
          - Construct a texture pyramid
            - At base goes original texture
            - Next level up: four texels averaged to one
            - Until one texel for whole texture
        - Size of pixel in texture space
          - Reverse project the corners of each pixel onto the pixel
          - Use a quadrilateral to connect the result, calculate area
        - Calculate color from pyramid
          - Find the level in which texels are just larger than the pixel area
          - Find the level in which texels are just smaller than the pixel area
          - Perform trilinear interpolation
      - Critique
        - +: Fast
        - +: No speckling, visually "okay"
        - -: Filtering is approximate, detail overblurred
  - Bump mapping (Blinn 78)
    - Problem
      - We want the surface to look more bumpy
      - Don’t want more polys
      - Texture mapping doesn’t cut it
    - Basic approach
      - Define a height field across the 2D parameter space
      - Use the defined surface to perturb model surface normals
      - Render with these normals
      - Can use Phong shading to make this happen inside a poly
    - Critique
      - + works!
      - - but not at silhouettes
      - - slow until pixel shaders
    - Variants
      - Normal mapping
        - Precalculate normal perturbations

- Store in “texture”: RGB = [XYZ]
  - Faster, but only possible recently
- Displacement mapping
  - Perturb the surface itself
  - + Get silhouettes
  - - But use lots of geometry

Environment mapping (Blinn & Newell 76)

- aka reflection mapping
- Problem
  - Want to render reflective objects
  - Need speed: can’t ray trace
- Basic approach
  - Looking out from the object, take pictures of surroundings
  - Map these pictures onto a cube (or a sphere, or…)
  - Map the cube onto the object
  - At render time, index into the pictures using these mappings
- Critique
  - +: works, sorta!
  - +: it’s interactive!
  - -: doesn’t work well if the reflective object is big in env’t
  - -: doesn’t handle concavities in object
  - -: sampling problems at mapping seems, singularities

Light mapping

- Problem
  - Want lighting with subpolygon accuracy
  - But want interactivity
  - Phong shading is too slow, not accurate enough
- Basic approach
  - Precompute lighting
  - Store results in a texture map
  - Combine with other textures
    - Combination can be precomputed or dynamic
- Critique
  - + Works well
  - + Its interactive
  - - only works for static, view independent lighting (generally)

Multipass texturing

- Problem
  - Want to combine several mapping techniques dynamically
  - Want interactivity
- Basic approach
  - Could render several times, blend in image
  - To speed this up, we offer hardware support
  - This enables blending of several textures in hardware

Advanced ideas

- Texture synthesis
  - Algorithms exist that generate texture images
  - Synthesis that depends on the surface
- Parameterizations
  - Mapping arbitrary 3D surfaces into 2D textures is hard
  - Lapped textures offers a solution
- Surface geometry
  - Adding fine geometry to the surface
  - E.g. hair, other protrusions