Bump Mapping:

a) Let $p(u,v)$ be a point on a parametric surface

then the normal $\vec{n} = \dfrac{\vec{P}_u \times \vec{P}_v}{|\vec{P}_u \times \vec{P}_v|}$ where $\vec{P}_u = \begin{bmatrix} \frac{\partial x}{\partial u} \\ \frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial u} \end{bmatrix}$ and

$\vec{P}_v = \begin{bmatrix} \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial v} \end{bmatrix}$ [partial derivatives lying in the tangent plane @ point P]

b) displace the surface in the normal direction by $d(u,v)$

then $\vec{p}' = \vec{p} + d(u,v)\,\vec{n}$ but we don't want the new pos, just the new normal: $\vec{n}' = \dfrac{\vec{P}_u' \times \vec{P}_v'}{|\vec{P}_u' \times \vec{P}_v'|}$

$$\vec{P}_u' = \vec{P}_u + \frac{\partial d}{\partial u}\vec{n} + d(u,v)\vec{n}_u \quad , \quad \vec{P}_v' = \vec{P}_v + \frac{\partial d}{\partial v}\vec{n} + d(u,v)\vec{n}_v$$

(where $\vec{n}_u$ & $\vec{n}_v$ are the u & v axes (binormal & tangent)]

we can precompute & store arrays of $\dfrac{\partial d}{\partial u}, \dfrac{\partial d}{\partial v}$

Displacement mapping:

- better: silhouette is correct
- worse: requires many triangles, ability to edit geometry in pipeline (vertex shaders)

Environment Mapping:

Sphere - $\vec{b} = -\vec{c} + \frac{2(\vec{c}\cdot\vec{n})\vec{n}}{\|\vec{c}\|^2}$, if we don't have the normal, assume object is a sphere (humans are bad at calculating reflections)

- once we have the vector $\vec{b}$, we can map it on to the sphere using $(\theta, \phi) = (\arccos(z), \operatorname{atan2}(y, x))$ then

$$(u, v) = \left(\frac{\phi}{2\pi}, \frac{\pi - \theta}{\pi}\right)$$

[standard spherical mapping, Shirley 11.2

Pro/Con:

- sphere: pro - Fast, directly supported by OpenGL
  - con - need a new sphere for each view position (not really: people are easy to fool)
    - speckle artifacts
- cube: pro - easy to change orientation for shifting viewpoint
  - can render scene from various directions to give real reflections
  - con - needs more memory, ability to access 6 textures at once

- Glass demo: 2 pass rendering:
  1) render white teapot to FBO
  2) distort white teapot with Fresnel eqns, (if behind glass), apply env. mapping

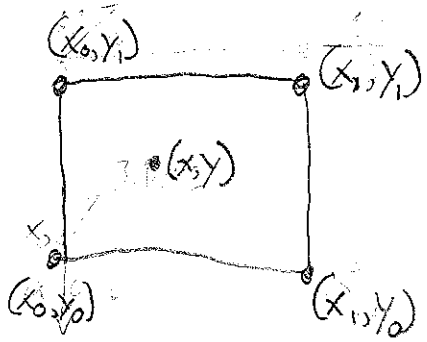
- Perlin Noise:

  - calculate which gradient vector to use: hash[hash[hash[LX]]+LY]+LZ

    where hash is a 256 element permutation, repeated twice

  - calculate ease function for fractional parts of x, y, z
    - ease function designed to have zero 1st & 2nd derivative at 0 & 1
  - average all 8 vector contributions using dot products

- 2D example:



where $x_0 = \lfloor x \rfloor$ $x_1 = x_0 + 1$
$y_0 = \lfloor y \rfloor$ $y_1 = y_1 + 1$

$$s = g(x_0,y_0)\cdot((x,y)-(x_0,y_0))$$
$$t = g(x_1,y_0)\cdot((x,y)-(x_1,y_0))$$
$$u = g(x_0,y_1)\cdot((x,y)-(x_0,y_1))$$
$$v = g(x_1,y_1)\cdot((x,y)-(x_1,y_1))$$

$$S_x = 6(x-x_0)^5 - 15(x-x_0)^4 + 10(x-x_0)^3$$
$$S_y = 6(y-y_0)^5 - 15(y-y_0)^4 + 10(y-y_0)^3$$

$$a = s + S_x(t-s)$$
$$b = u + S_x(v-u)$$
$$out = a + S_y(b-a)$$

- Concentration for a particular morphogen given by $C(x,y)$
- reaction-diffusion equation given by $\dot{C} = a^2 \nabla^2 C - bC + R$ where $\dot{C}$ is the time derivative of C
- $\nabla^2 C$ is the laplacian, defined by $\nabla^2 C = \frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}$

approximated by the finite difference $\frac{\partial^2 C}{\partial x^2} \approx \frac{C_{i+1,j} + C_{i-1,j} - 2C_{i,j}}{h^2}$

where h is the distance between samples

- the discrete laplacian can also be expressed as the convolution of the concentration array with the 3×3 mask: $\frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

multiplying in the a^2 and adding the $-bC$ term gives the mask $M = \frac{1}{h^2}\begin{bmatrix} 0 & a^2 & 0 \\ a^2 & (-4h^2 - b)a^2 & a^2 \\ 0 & a^2 & 0 \end{bmatrix}$

then the R-D equation becomes $\dot{C} = M * C + R$ (* = discrete convolution)

we can then integrate this w/ Euler's method to get $C_{t+\Delta t} = \Delta_t (M * C_t + R)$

- we can add more complications if we want anisotropic, or space-varying diffusion
- also need to do smarter things (multi-grid method or Gaussian convolution) to solve quickly