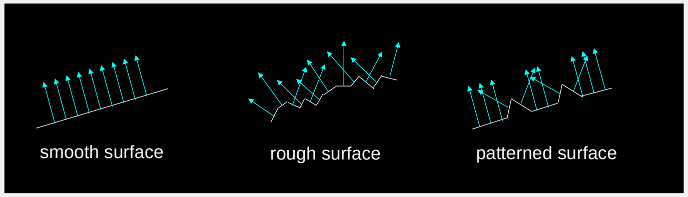
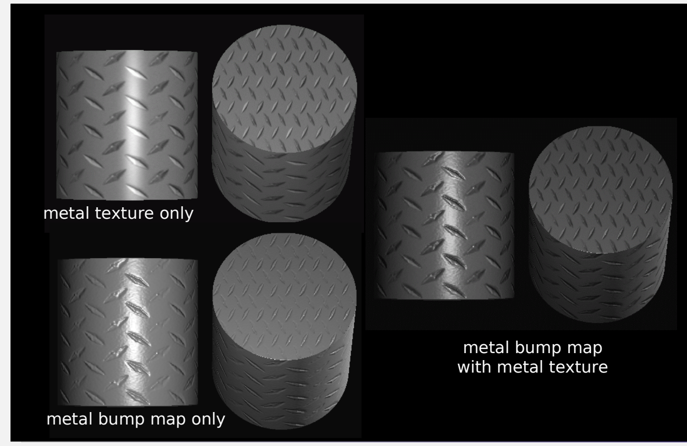
Bump Mapping

the idea: on a rough or patterned surface, surface normals aren't all parallel

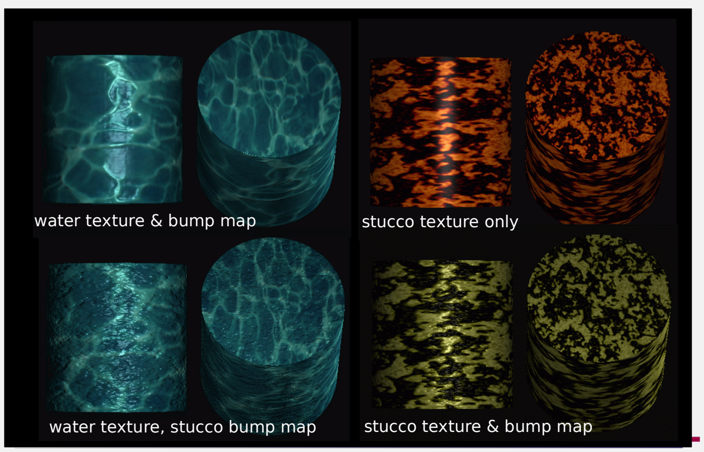


- it is too expensive to actually model a complex surface with polygons, but the effect can be approximated by modeling the smooth surface but perturbing the normals in lighting calculations
- can use textures (as a bump map) to define how to perturb the normals

Examples



Examples

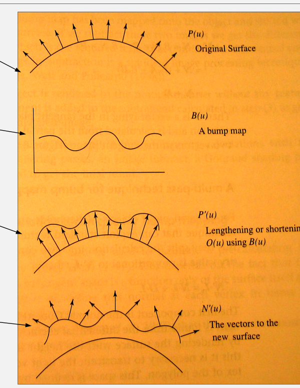


Bump Mapping

idea –

- perturb surface point (the bump)
- compute (an approximate) normal for the perturbed point

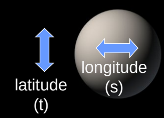
- define texture coordinates (u,v) for each point P on the surface
- define bump map B(u,v)
  (maps points to displacements)
- at each point P on the surface, displace by B(u,v) along the normal at P
  P' = P + B(u,v) N
- compute normal N' for P'
- use N' in lighting and other computations



Defining Texture Coordinates

- can use generation techniques previously discussed
- for shapes where there is a convenient parametric representation, use the shrinkwrap approach

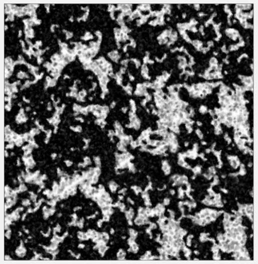
$$x = r\cos(lat)\sin(long)$$
$$y = r\sin(lat)$$
$$z = r\cos(lat)\cos(long)$$



  - for surface point (x,y,z)
    - solve for (r,lat,long)
    - map long → u, lat → v

Defining the Bump Map

- to reduce computations, obtain bump values via table lookup instead of evaluating a function
- B(u,v) is typically defined by 2D *height field* obtained from a grayscale bitmap image
  - values 0 to 255 → map to range [-128,128]



Computing Perturbed Normals

- displaced point

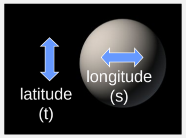
$$P' = P + B(u,v)\,N$$

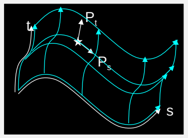
- approximation to displaced normal
  [Blinn 1978]

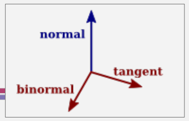
$$N' = N + B_u\,(N \times P_t) - B_v\,(N \times P_s)$$

(normalize before use)

  - P_s and P_t are the surface tangents along the parameterization axes
    - also known as *tangent* (P_s) and *binormal* (P_t)
  - B_u and B_v are the partial derivatives of B(u,v) with respect to u and v, respectively

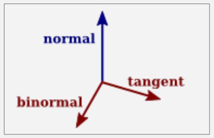






Computing the Tangent and Binormal

- book's discussion assumes `normal` and `tangent` (P_s) are defined as part of the surface
  - suitable when you don't have parametric equations defining the surface (e.g. poly mesh)
- compute `binormal` (P_t) as `normal` × `tangent`

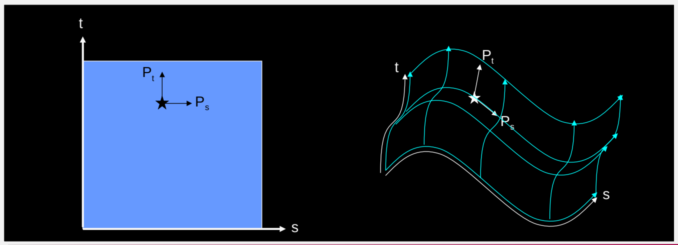



```
vec3 normal = normalize( v_normal );
vec3 tangent = normalize( v_tangent );
vec3 binormal = cross(normal,tangent);
```

Computing the Tangent and Binormal

For surfaces defined by parametric equations –

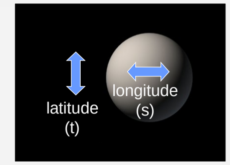
- P_s is partial derivative of P with respect to s
- P_t is partial derivative of P with respect to t
- surface normal at P is the cross product $P_s \times P_t$



Sphere Example

- parametric definition

  x = r cos(lat) sin(long)
  y = r sin(lat)
  z = r cos(lat) cos(long)



- partial derivatives

$$P_s \begin{cases} x' = r\cos(lat)\cos(long) \\ y' = 0 \\ z' = -r\cos(lat)\sin(long) \end{cases} \qquad P_t \begin{cases} x' = -r\sin(lat)\sin(long) \\ y' = r\cos(lat) \\ z' = -r\sin(lat)\cos(long) \end{cases}$$

- $N = P_s \times P_t$

$$P_s \times P_t = \begin{bmatrix} r^2\cos^2(lat)\sin(long) \\ r^2\cos(lat)\sin(lat)\cos^2(long) + r^2\cos(lat)\sin(lat)\sin^2(long) \\ r^2\cos^2(lat)\cos(long) \end{bmatrix} = r\cos(lat) \begin{bmatrix} r\cos(lat)\sin(long) \\ r\sin(lat) \\ r\cos(lat)\cos(long) \end{bmatrix}$$

  – which points in the same direction as (x,y,z)

Computing B_u and B_v

- approximate derivatives B_u and B_v by looking at differences between neighboring entries in bitmap
- obtain B_u by convolving the bump map image with

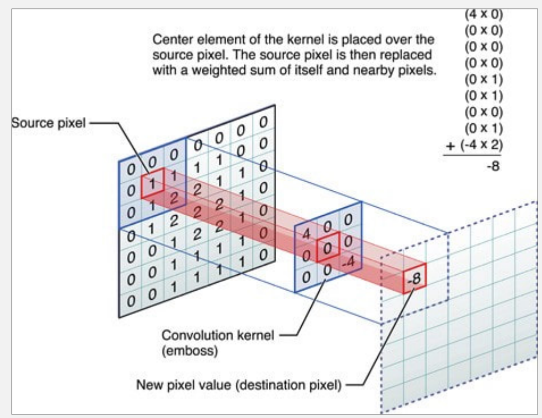
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

- obtain B_v by convolving the bump map image with

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

scale result by range of values in bitmap

Convolution



Computing B_u and B_v

note: this is opposite what is in the book – this version means that white in the bump map corresponds to a larger distortion outwards

- the book uses a simpler convolution
- obtain B_u by convolving the bump map image with $\begin{bmatrix} 0 & 1 & -1 \end{bmatrix}$
- obtain B_v by convolving the bump map image with $\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$

```
float bm0 = texture2D( u_bumpmap, v_texCoords ).r;
float bmUp = texture2D( u_bumpmap, v_texCoords + vec2(0.0, 1.0/u_bumpmapSize.y) ).r;
float bmRight = texture2D( u_bumpmap, v_texCoords + vec2(1.0/u_bumpmapSize.x, 0.0) ).r;
vec3 bumpVector = (bm0-bmRight )*tangent + (bm0-bmUp )*binormal;
normal += u_bumpmapStrength*bumpVector;
```

one pixel up/right

`.r` = red component of color (same as g, b for grayscale image)

`tangent` and `binormal` are ($N \times P_t$) and ($N \times P_s$), respectively

scale result by range of values in bitmap
if this is a constant range, can scale by desired strength

note: normal is still in OC – must convert to EC before using in lighting equation

Implementing Bump Mapping

- to compute the perturbed normal N' for OC point (x,y,z)
  - map OC (x,y,z) to TC (u,v)
  - scale TC (u,v) to BC (u',v') – i.e. apply texture transform
  - compute B_u and B_v for (u',v')
  - compute tangent $N \times P_t$ and binormal $N \times P_s$ if needed
  - compute N' = N + B_u `tangent` - B_v `binormal`
- use N' instead of regular surface normal for illumination and other lighting-related calculations

Examples