# Bump Mapping

1. Normal Mapping
2. Height Mapping
3. Parallax Mapping
4. Procedural Normal Mapping

## Bump Mapping

- Bump mapping
  - A cheap computer graphics technique to model bumpy surface without changing the underlying surface model
  - Based on the fact
    - what we see is the amount of light arriving at our eyes
    - perturb the surface normal at a point on the surface will change the amount of light arriving at eyes from that point
      - diffuse light, specular light, …

## Bump Mapping Ideas

- Normal mapping
  - Surface normal is read directly from a normal map
- Parallax mapping
  - Use two maps: a height map and a normal map
    - Height map is used for parallax compensation
- Height mapping
  - Surface normal is found from a height map
- Procedural bump mapping
- Displacement mapping
  - The actual geometric position of points over the textured surface are *displaced*
- Relief mapping
  - A much finer bump mapping technique
    - Supporting self-occlusion, self-shadowing, view-motion parallax, and silhouettes

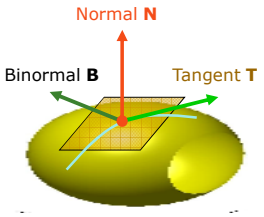
## Bump Mapping by Perturbing the Normal

## Transform into Tangent Space

- Bumpy information is often represented in tangent space
- To do bump mapping, relevant position and directional information for computing illumination quantities need to be transformed into the same space
  - Usually in tangent space

## Transform into Tangent Space

$$\begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{pmatrix}$$




This matrix depends on how the local coordinate system is defined for The underlying geometric model

## Transform into View Space

```
… …
varying vec2 Texcoord;
varying vec3 ViewDirInTangent ;
varying vec3 LightDirInTangent ;

attribute vec3 rm_Binormal;
attribute vec3 rm_Tangent;

void main( void )
{
    … …
    //Find vectors of view direction, light direction:
    vec3 ViewDir  =  - PView.xyz;
    vec3 LightDir = LightPos.xyz - PView.xyz;
```

## Transform into Tangent Space

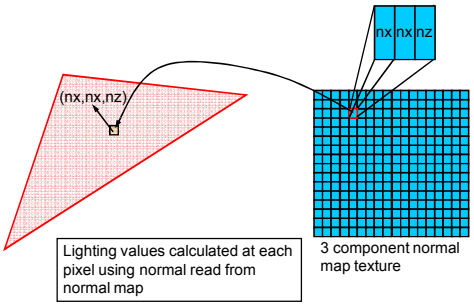
```
//1. Transform Normal, Binormal and Tangent vectors into view space:
vec3 fvNormal      = gl_NormalMatrix * gl_Normal;
vec3 fvBinormal    = gl_NormalMatrix * rm_Binormal;
vec3 fvTangent     = gl_NormalMatrix * rm_Tangent;

//2. Construct transformation matrix:
mat3 View2Tangent
     = mat3(fvTangent.x, fvBinormal.x, fvNormal.x,  //first col
            fvTangent.y, fvBinormal.y, fvNormal.y,  //secod col
            fvTangent.z, fvBinormal.z, fvNormal.z); //third col

//3. Transform Normal, Binormal and Tangent vectors into tangent space:
ViewDirInTangent =  View2Tangent *  ViewDirInViewSp;
LightDirInTangent =  View2Tangent *  LightDirInViewSp;
```

## 1. Normal Mapping

—Read normal from a normal map

- **Normal maps** are images that store normals directly in RGB values
  - The RGB values of each texel in the normal map represent the x,y,z components of the normalized normal vector at the vertex associated with the texel
- Instead of using interpolated vertex normals to compute the light colour, the normals from the normal map are used

## Normal Map




## How to Use the Normal Map

- Colour values in a texture are typically constrained to [0, 1]
- Since the components of a normal vector, when normalized, correspond to [-1, 1], they must be compressed into [0, 1] when they are stored in colour
- When a bump texture is loaded for the purpose of bump mapping, it must be de-compressed

  Colour2Normal = 2*Normal2Colour - 1

## Bumpy Effect Using a Normal Map

```
vec3 NormalAsCol  = texture2D(bumpMap, Texcoord ).xyz;

vec3 NormalFrCol = normalize( 2.0 * NormalAsCol - 1.0 );
```

## 2. Bump Mapping
### — Computing normal from a height map

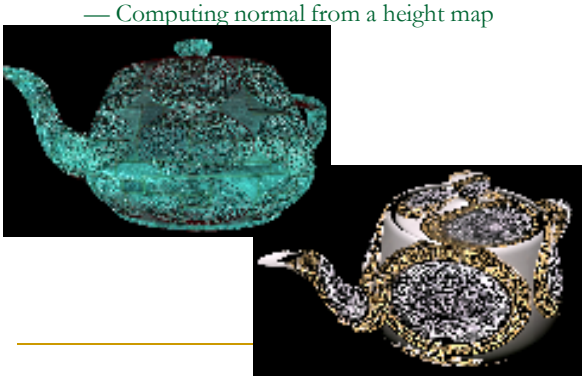
- Height map
  - A texture storing the surface height information
  - Normally created in the tangent space
  - Normal vectors at each pixel is then calculated from the height map
  - Implement similarly to procedural bump mapping

## Bump Mapping
### — Computing normal from a height map

- Scale the texture coordinates:
  float x = BumpDensity * Texcoord.x,
  y=BumpDensity * Texcoord.y;
- Use a height map to represent the height function
  vec3 F = texture2D( heightMap, Texcoord ). z;
- Find the partial derivatives of the height function:
  float dx = 0.01; float dy = 0.01;
  float Fx=texture2D(heightMap, vec2(x+dx,y) ).z;
  float Fy=texture2D(heightMap, vec2(x,y+dy) ).z;
  float dFx=(Fx-F)/dx;
  float dFy=(Fy-F)/dy;

  vec3 Normal = normalize( vec3(-dx, -dy, 1.0) );

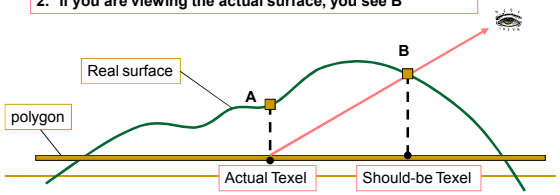
## Bump Mapping
### — Computing normal from a height map



## Problem of Bump Mapping

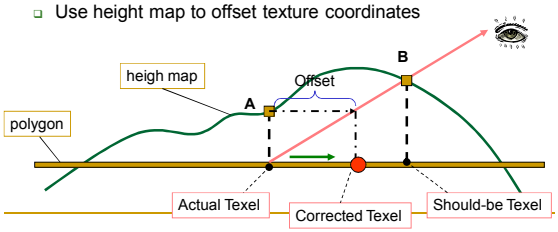
Cannot represent view-dependent unevenness

**What do you see?**
1. **If you are viewing texture mapped polygon, you see A**
2. **If you are viewing the actual surface, you see B**

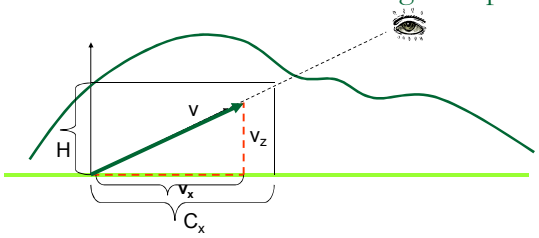



## 3. Parallax mapping

- Q: How can we improve traditional bump mapping such that what we see is close to the real situation?
- Solution:
  - Use height map to offset texture coordinates




## How to find the offset for a height map?

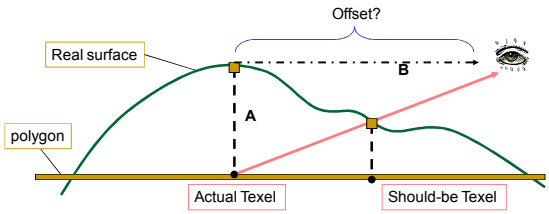



$$\frac{H}{C_x}=\frac{v_z}{v_x}.\quad \text{So}\quad C_x=\frac{v_x}{v_z}H$$

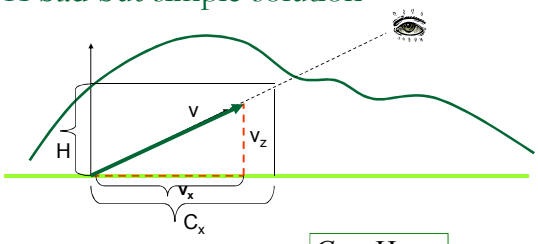
$$\frac{H}{C_y}=\frac{v_z}{v_y}.\quad \text{So}\quad C_y=\frac{v_y}{v_z}H$$

## Problem?

As the viewing angle becomes more shallow, offset values Approach infinity!




## A bad but simple solution

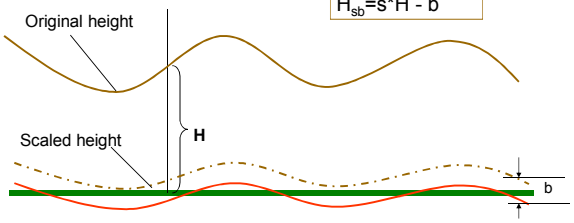



$$C_x = \mathrm{H} \times v_x$$
$$C_x = \mathrm{H} \times v_y$$

## Use modified height

Height value **H** must be scaled and biased to account for the surface being simulated

Replace h with: $H_{sb}$=s*H - b




## Pixel shader: Input

```
uniform vec4 AmbientLight;
… …

uniform sampler2D ColorMap;
uniform sampler2D NormalMap;
uniform sampler2D HeightMap;

varying vec2 Texcoord;
varying vec3 ViewDirection;
varying vec3 LightDirection;

uniform float scale;
uniform float bias0;
```

## Pixel shader: main( )

```
vec3  L = normalize( LightDirection );
vec3  V = normalize( ViewDirection );

// fetch height map
float Height = texture2D( HeightMap, Texcoord ).x;
Height = scale* Height - bias0;

// Correct the texture coordinates
vec2 TexCorrected  = Texcoord
                   + Height * V .xy;
```

## Pixel shader (cont. )

```
//fetch texture color
vec4  BaseColor  =
            texture2D( ColorMap, TexCorrected );

// fetch normal vector
vec3 Normal =2.0 * texture2D( BumpMap,
                        TexCorrected )  - 1.0;

// Compute light colour
… …
```