Computer Graphics (CS 543)
Lecture 10: Normal Maps, Parametrization, Tone Mapping

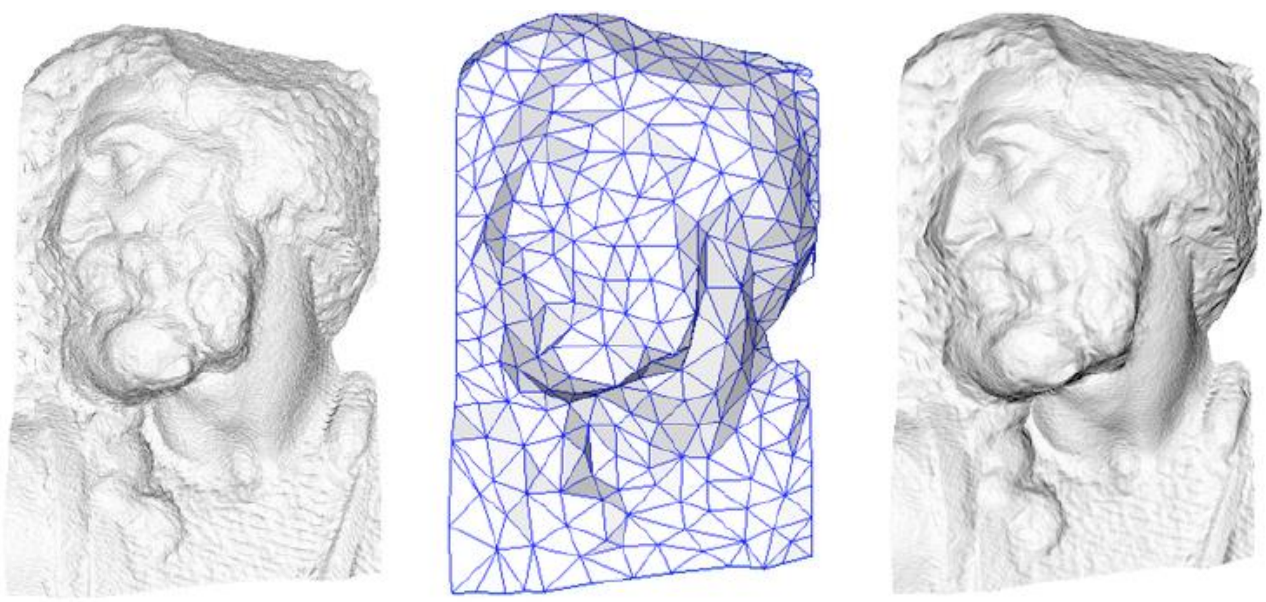
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Normal Mapping

- Store normals in texture
- Normals <x,y,z> stored in <r,g,b> values in texture
- Normal map may change a lot, simulate fine details
- Low rendering complexity method for making low-resolution geometry look like it’s much more detailed



original mesh
4M triangles

simplified mesh
500 triangles

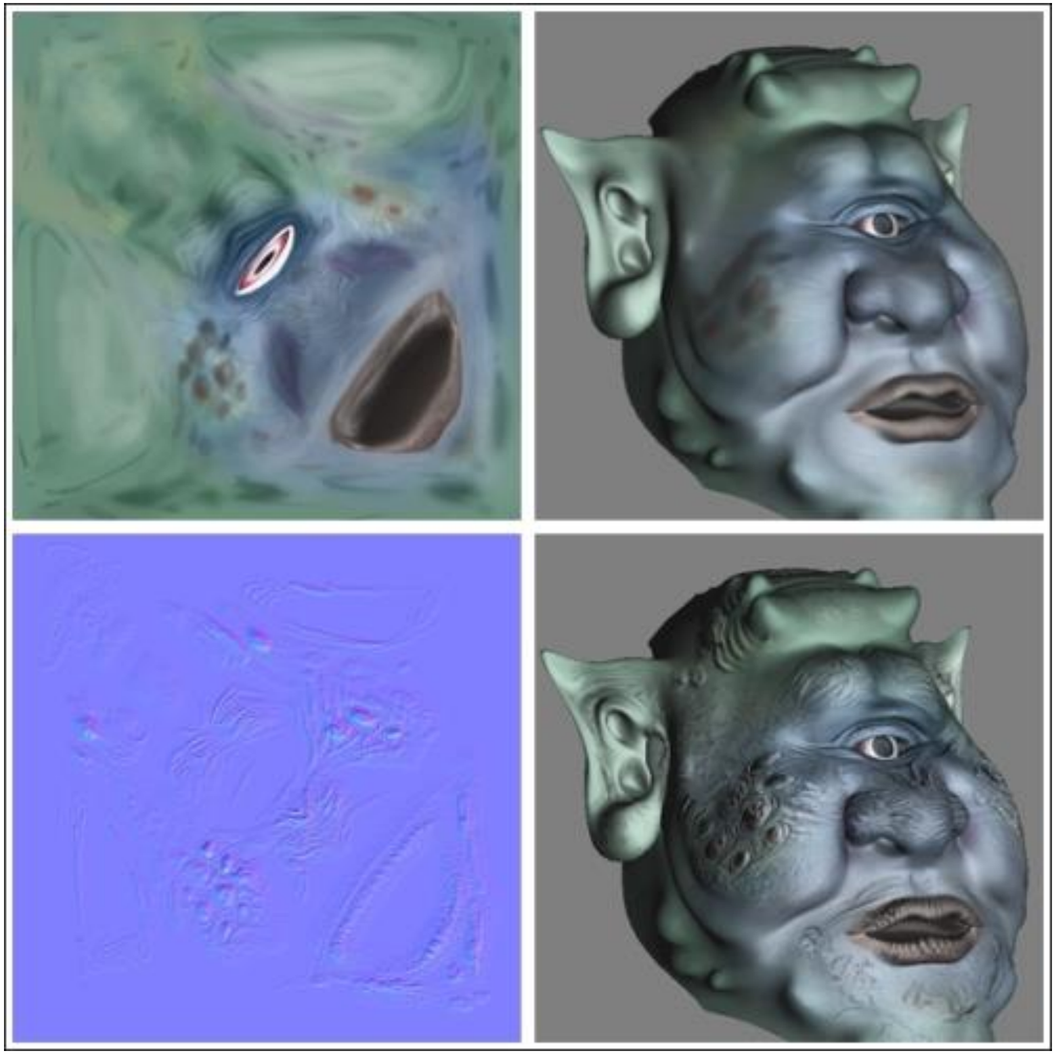
simplified mesh
and normal mapping
500 triangles

Normal Mapping Example: Ogre

OpenGL 4 Shading Language Cookbook (2nd edition) by David Wolff (pg 130)

Base color texture (used this in place of diffuse component)

Texture mapped Ogre (Uses mesh normals)

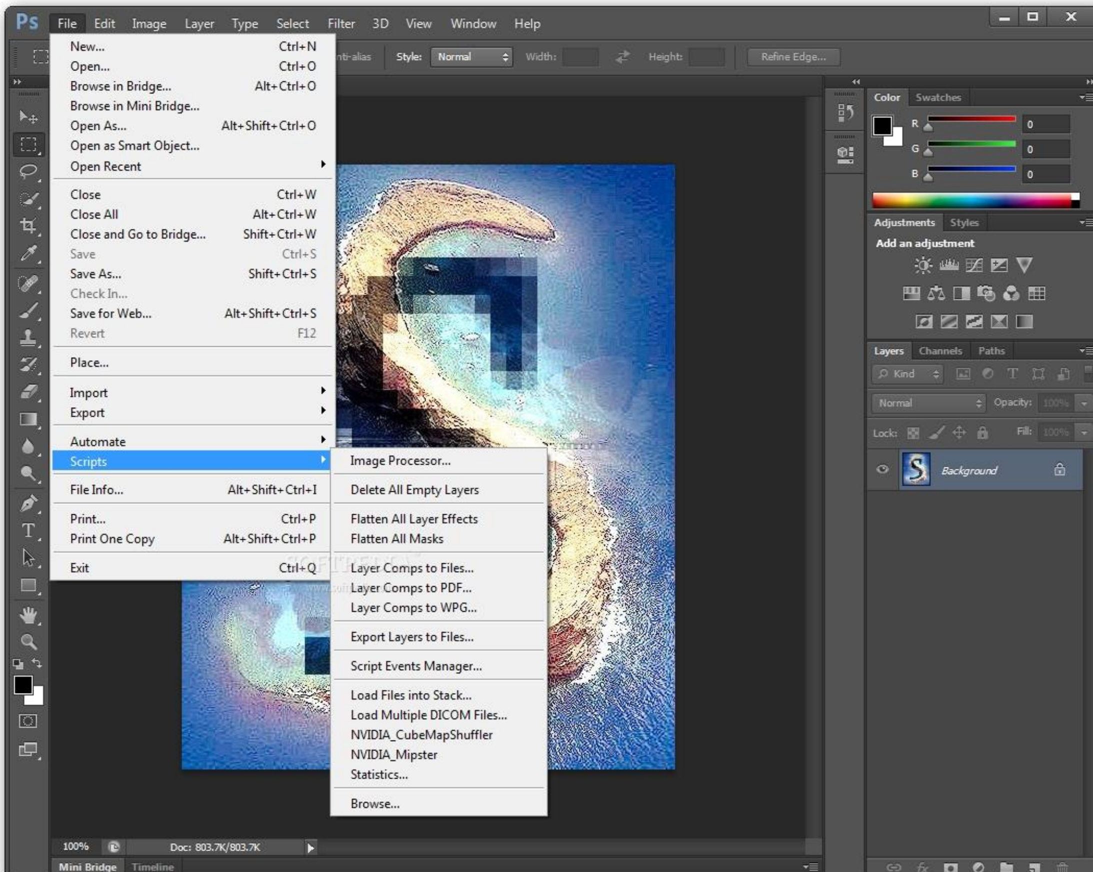


Normal texture map

Texture and normal mapped Ogre (Uses normal map to modify mesh normals)

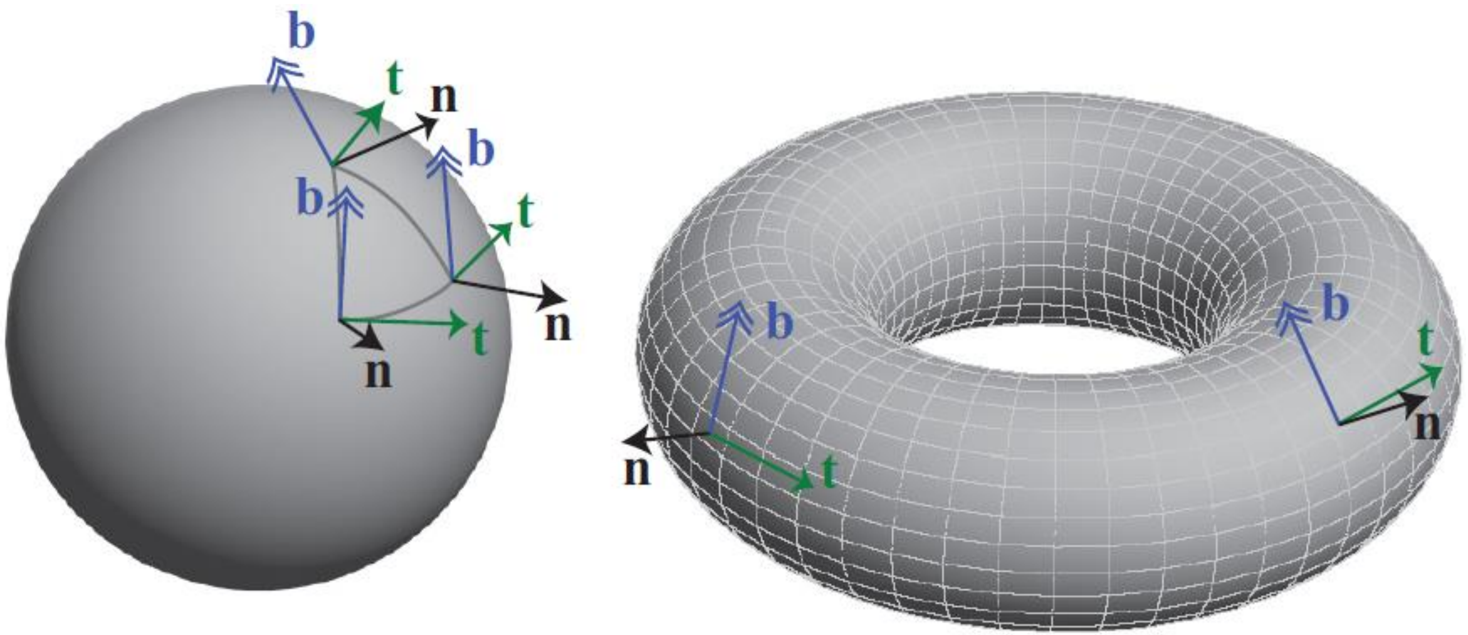
Creating Normal Maps

- Many tools for creating normal map
- E.g. Nvidia texture tools for Adobe photoshop
  - https://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop



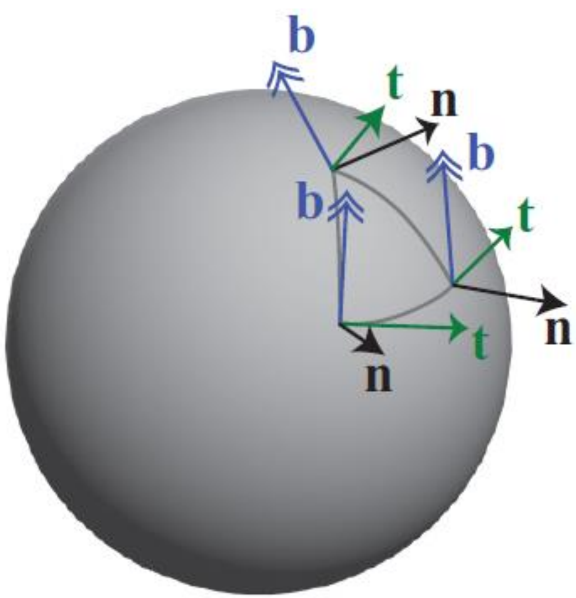
Tangent Space Vectors

- Normals in normal map stored in object local coord. frame (or tangent space)
- Object Local coordinate space? Axis positioned on surface of object (NOT global x,y,z)
- Need Tangent, normal and bi-tangent vectors at each vertex
  - z axis aligned with mesh normal at that point
  -

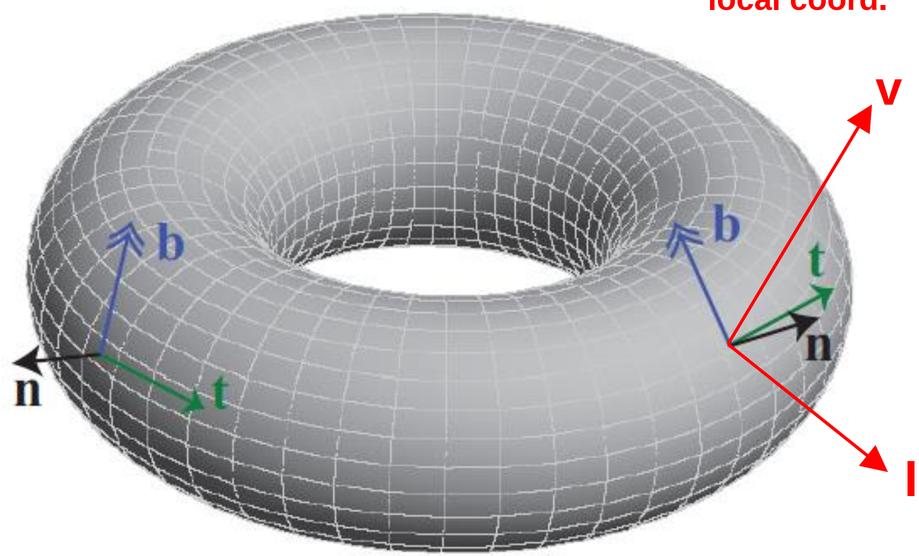


Tangent Space Vectors

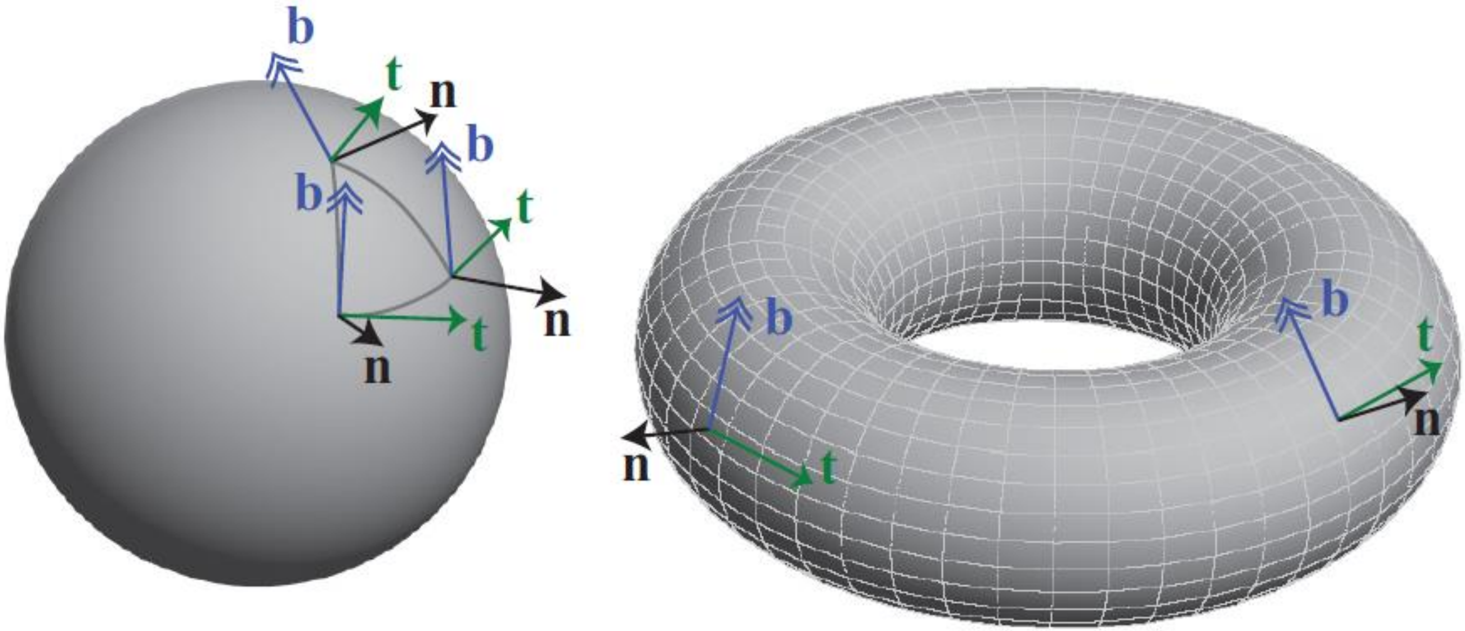
- Normals stored in texture includes mesh transformation + local deviation (e.g. bump)
- Reflection model must be evaluated in object's local coordinate (n, t, b)
- Need to transform view, light and normal vectors into object's local coordinate space



Need to transform l, v and n into object local coord.



Transforming V,L and N into Object's Local Coordinate Frame



- To transform a point ***P*** eye into a corresponding point ***S*** in object's local coordinate frame:

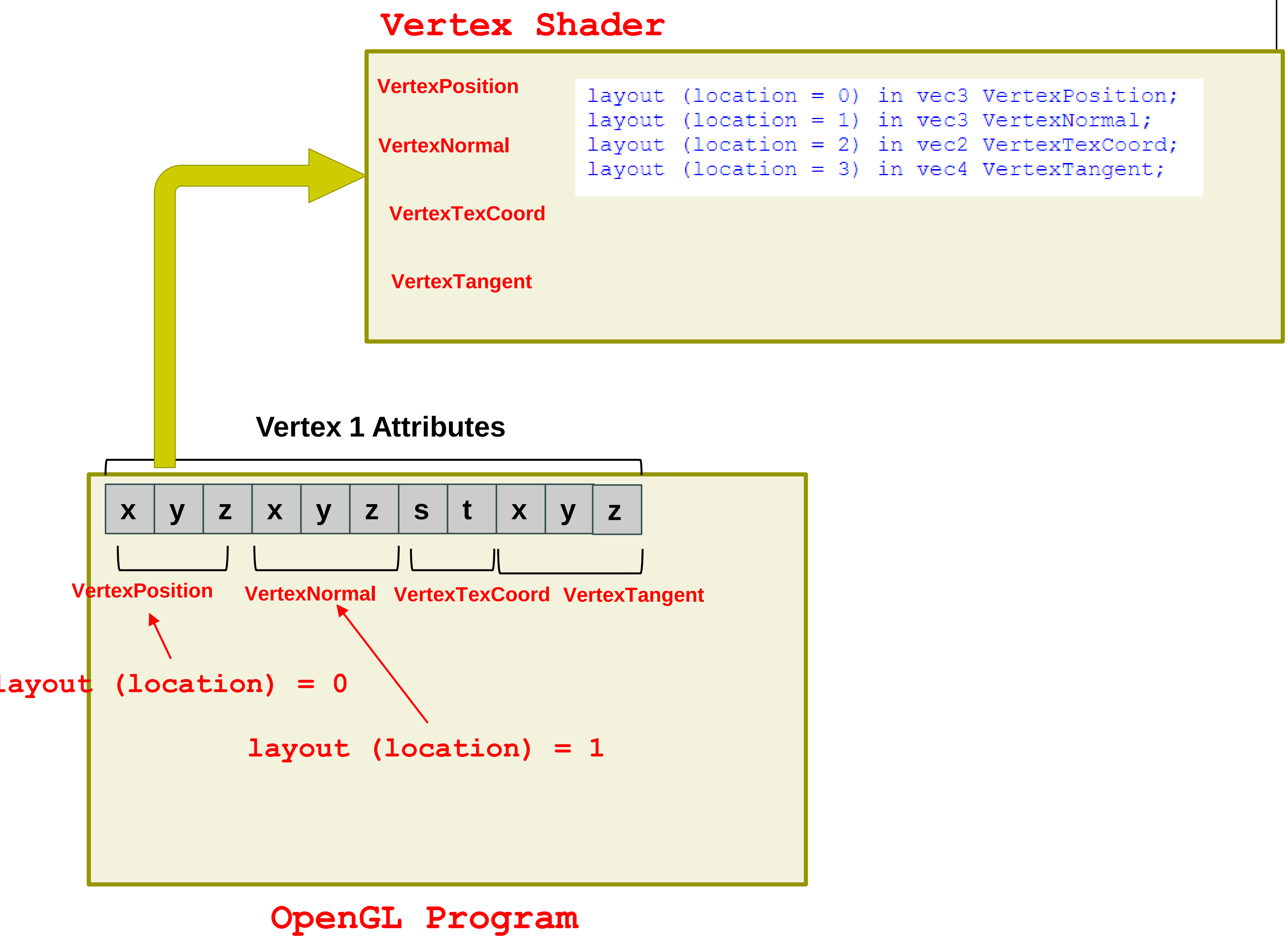
Point *S* in object's locatl coordinate frame →

$$\begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \\ n_x & n_y & n_z \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

← **Point *P* in eye coordinate frame**

Normal Mapping Example

OpenGL 4 Shading Language Cookbook (2nd edition) by David Wolff (pg 133)



Normal Mapping Example

OpenGL 4 Shading Language Cookbook (2nd edition) by David Wolff (pg 133)

Vertex Shader

```
layout (location = 0) in vec3 VertexPosition;
layout (location = 1) in vec3 VertexNormal;
layout (location = 2) in vec2 VertexTexCoord;
layout (location = 3) in vec4 VertexTangent;
```

....

```
uniform mat4 ModelViewMatrix;
uniform mat3 NormalMatrix;
uniform mat4 ProjectionMatrix;
uniform mat4 MVP;

void main()
{
  // Transform normal and tangent to eye space
  vec3 norm = normalize(NormalMatrix * VertexNormal);
  vec3 tang = normalize(NormalMatrix *
                        vec3(VertexTangent));
  // Compute the binormal
  vec3 binormal = normalize( cross( norm, tang ) ) *
  VertexTangent.w;
```

....

Transform normal and tangent to eye space

….

Compute bi-normal vector

```
// Matrix for transformation to tangent space
mat3 toObjectLocal = mat3(
tang.x, binormal.x, norm.x,
tang.y, binormal.y, norm.y,
tang.z, binormal.z, norm.z ) ;
```

Form matrix to convert from eye to local object coordinates

$$\begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \\ n_x & n_y & n_z \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$$

Normal Mapping Example

OpenGL 4 Shading Language Cookbook (2nd edition) by David Wolff (pg 133)

Vertex Shader

```
........................
  // Get the position in eye coordinates
  vec3 pos = vec3( ModelViewMatrix *
                   vec4(VertexPosition,1.0) );

  // Transform light dir. and view dir. to tangent space
  LightDir = normalize( toObjectLocal *
                        (Light.Position.xyz - pos) );
  ViewDir = toObjectLocal * normalize(-pos);

  // Pass along the texture coordinate
  TexCoord = VertexTexCoord;

  gl_Position = MVP * vec4(VertexPosition,1.0);
}
```

Get position in eye coordinates
….
Transform light and view directions to tangent space

Fragment Shader

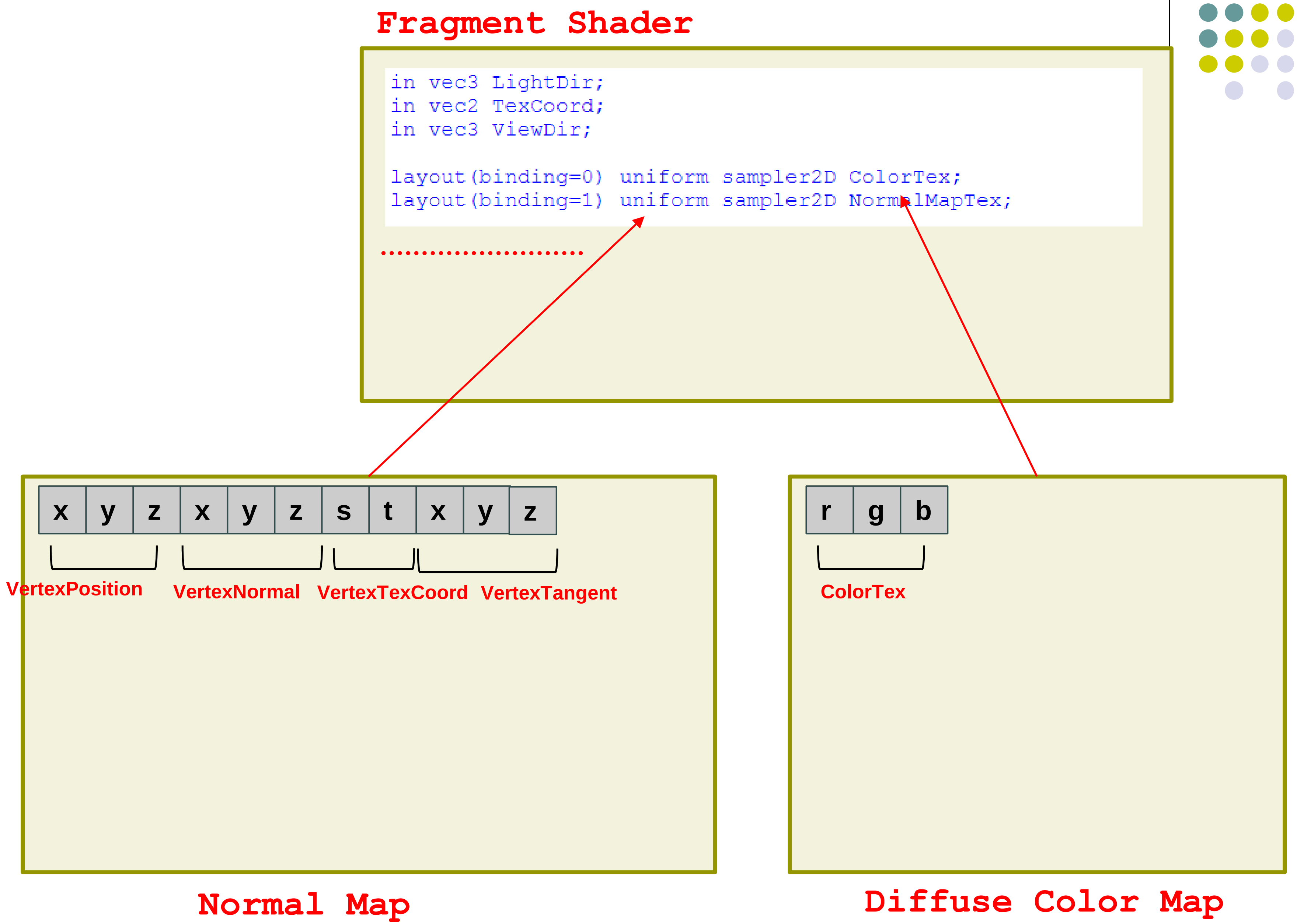
```
in vec3 LightDir;
in vec2 TexCoord;
in vec3 ViewDir;

layout(binding=0) uniform sampler2D ColorTex;
layout(binding=1) uniform sampler2D NormalMapTex;

........................
```

Receive Light, View directions and TexCoord set in vertex shader
……

Declare Normal and Color maps

Normal Mapping Example

OpenGL 4 Shading Language Cookbook (2nd edition) by David Wolff (pg 133)



Normal Mapping Example

OpenGL 4 Shading Language Cookbook (2nd edition) by David Wolff (pg 133)

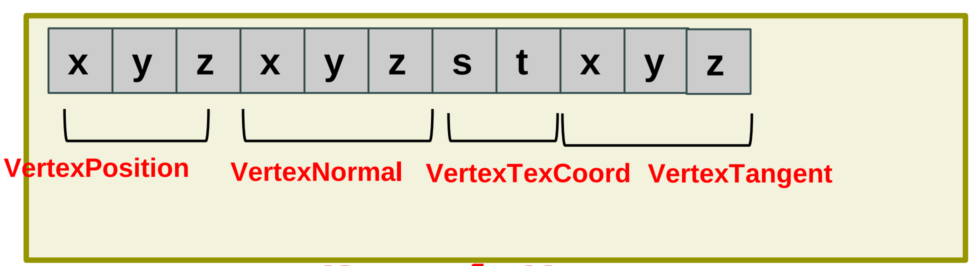
Fragment Shader

```
vec3 phongModel( vec3 norm, vec3 diffR ) {vec3 r = reflect( -LightDir, norm );
  vec3 ambient = Light.Intensity * Material.Ka;
  float sDotN = max( dot(LightDir, norm), 0.0 );
  vec3 diffuse = Light.Intensity * diffR * sDotN;

  vec3 spec = vec3(0.0);
  if( sDotN > 0.0 )
        spec = Light.Intensity * Material.Ks *
               pow( max( dot(r,ViewDir), 0.0 ),
                       Material.Shininess );

  return ambient + diffuse + spec;
}

void main() {
  // Lookup the normal from the normal map
  vec4 normal = 2.0 * texture( NormalMapTex, TexCoord ) -
                       1.0;

  // The color texture is used as the diff. reflectivity
  vec4 texColor = texture( ColorTex, TexCoord );

  FragColor = vec4( phongModel(normal.xyz, texColor.rgb),
                     1.0 );
}
```

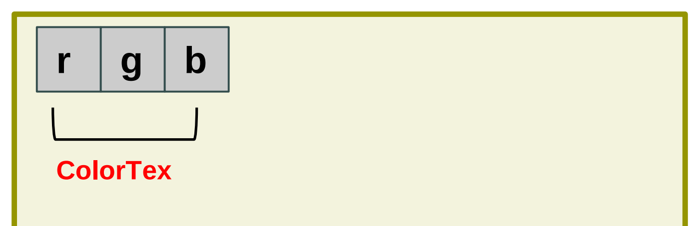
Function to compute Phong's lighting model

Look up normal from normal map

Look up diffuse coeff. from color texture

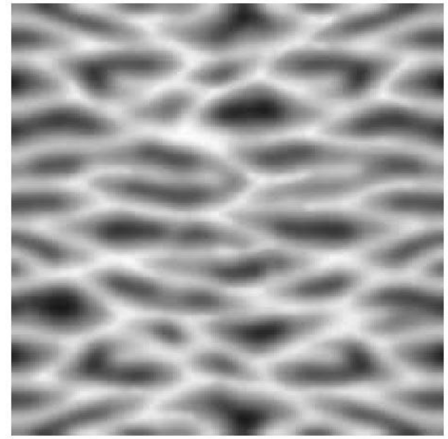


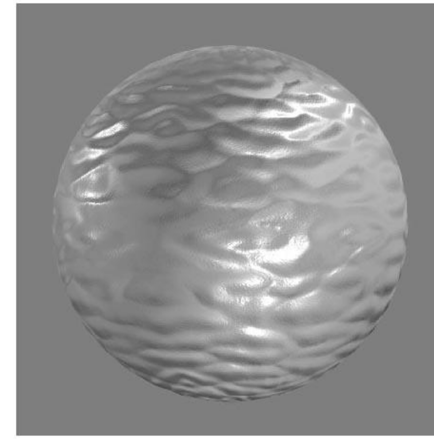
Normal Map



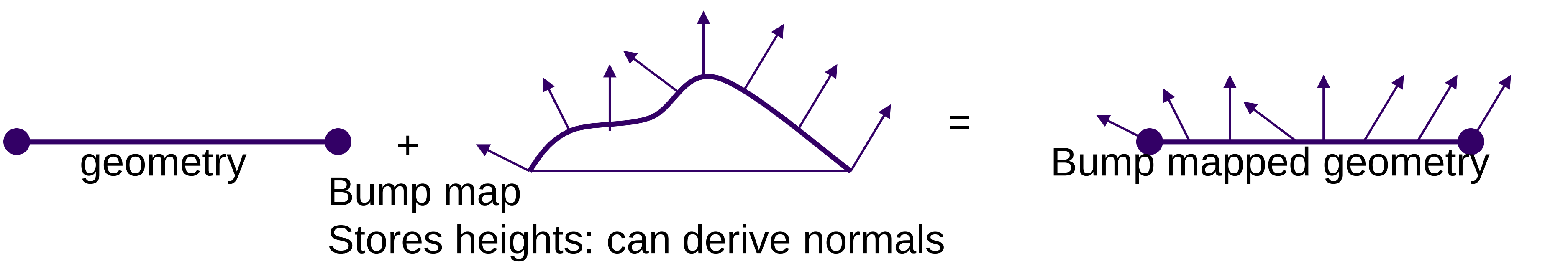
Diffuse Color Map

Bump mapping

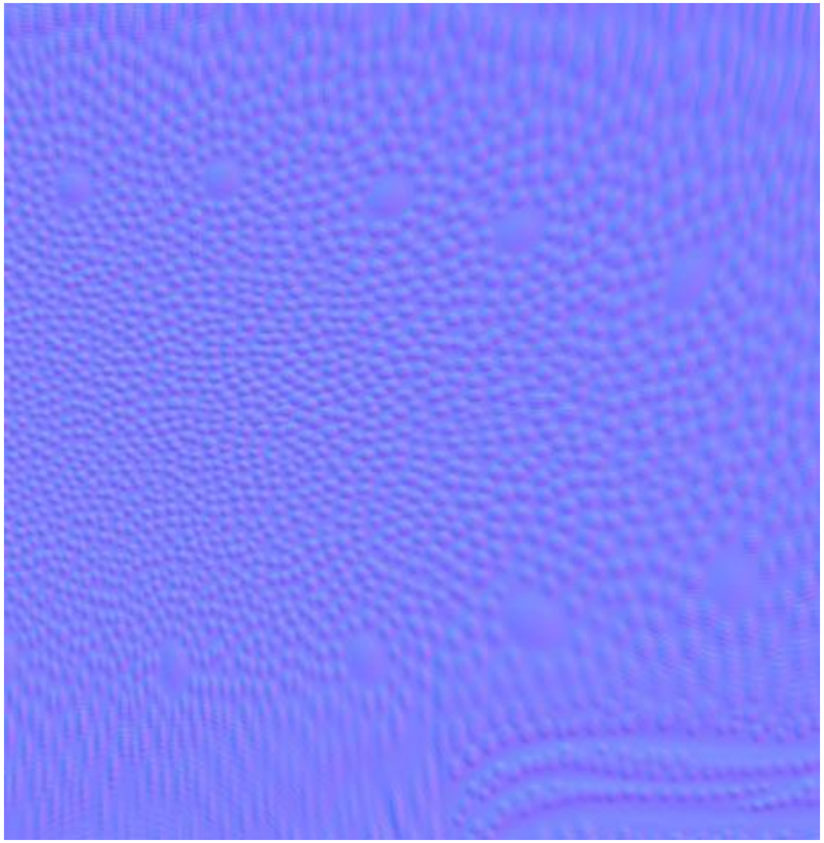




- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
  - Too expensive to model these geometrically
- Instead let a texture modify the normal at each pixel, and then use this normal to compute lighting

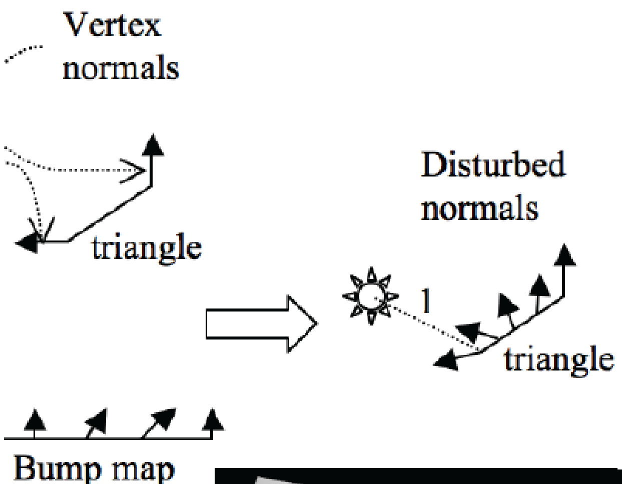


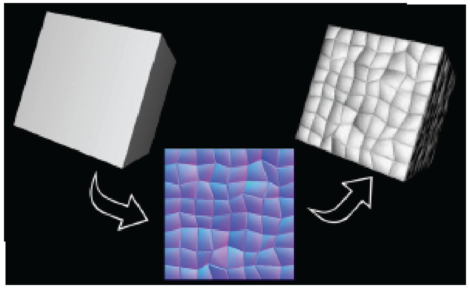
Bump mapping: examples



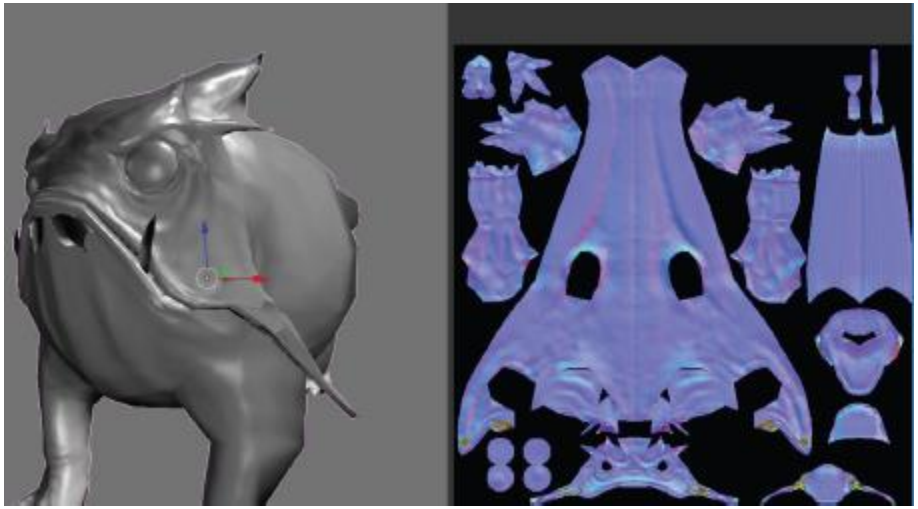
Bump Mapping Vs Normal Mapping

- **Bump mapping**
- (Normals $\mathbf{n}=(n_x, n_y, n_z)$ stored as *distortion of face orientation*. Same bump map can be tiled/repeated and reused for many faces)



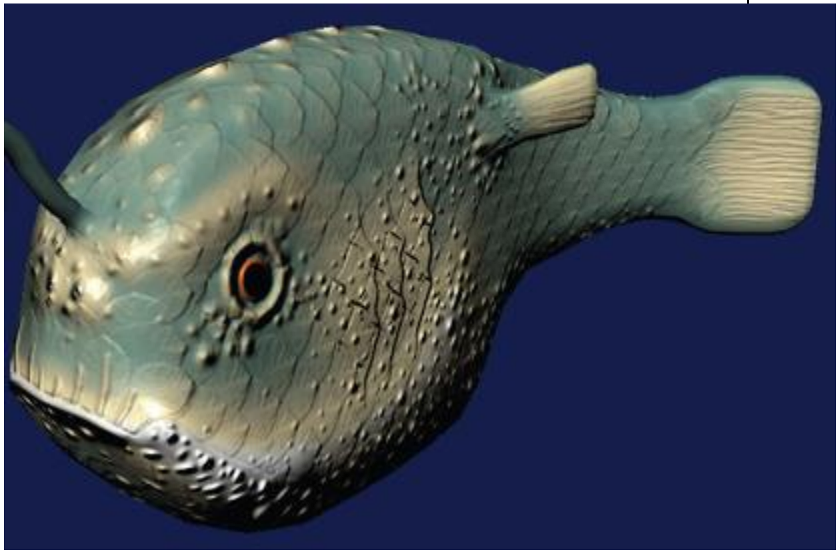


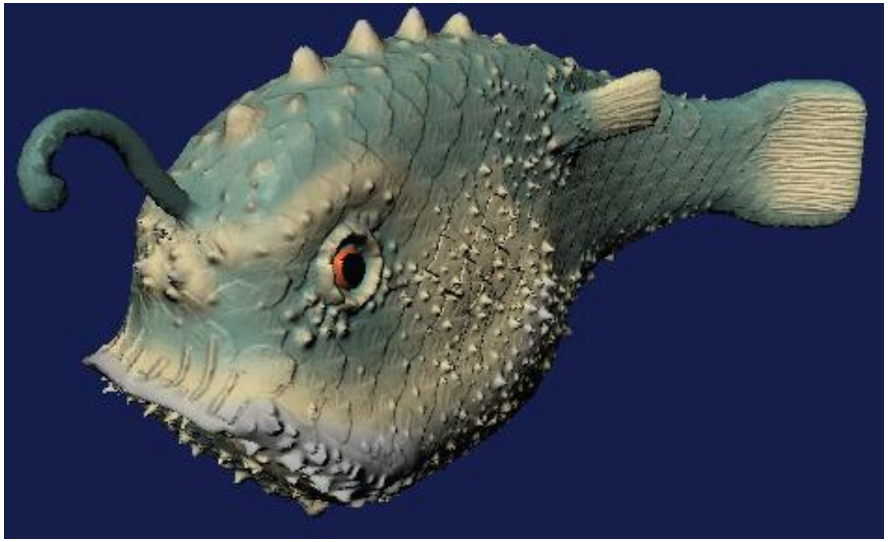
- **Normal mapping**
- Coordinates of normal (relative to tangent space) are encoded in color channels
- Normals stored include face orientation + plus distortion.)



Displacement Mapping

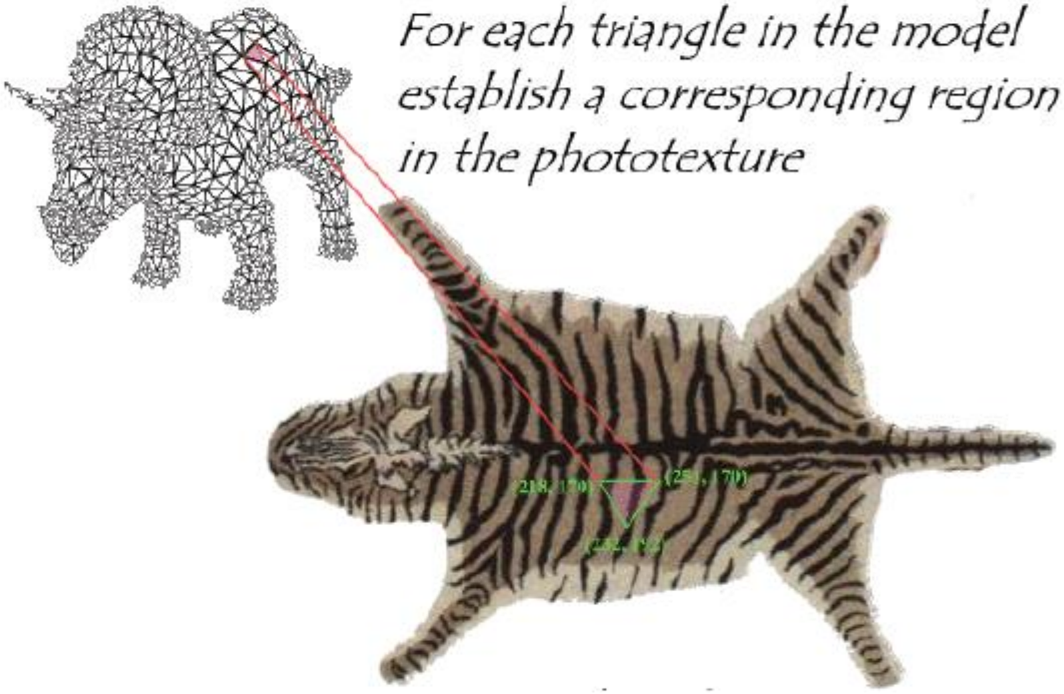
- Uses a map to displace the surface at each position
- Offsets the position per pixel or per vertex
  - Offsetting per vertex is easy in vertex shader
  - Offsetting per pixel is architecturally hard





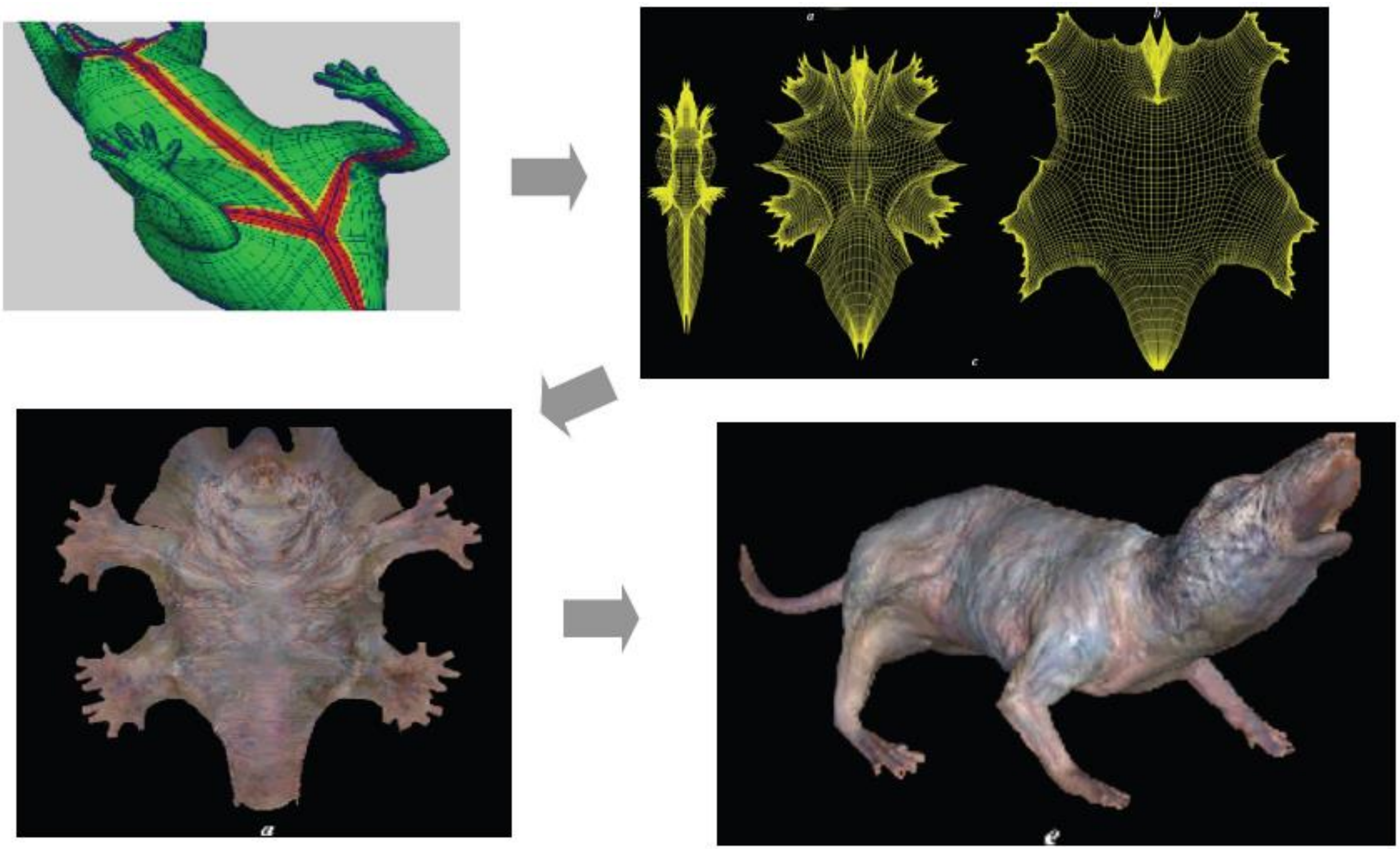
Hot Research Topic: Parametrization

- The concept is very simple: define a mapping from the surface to the plane



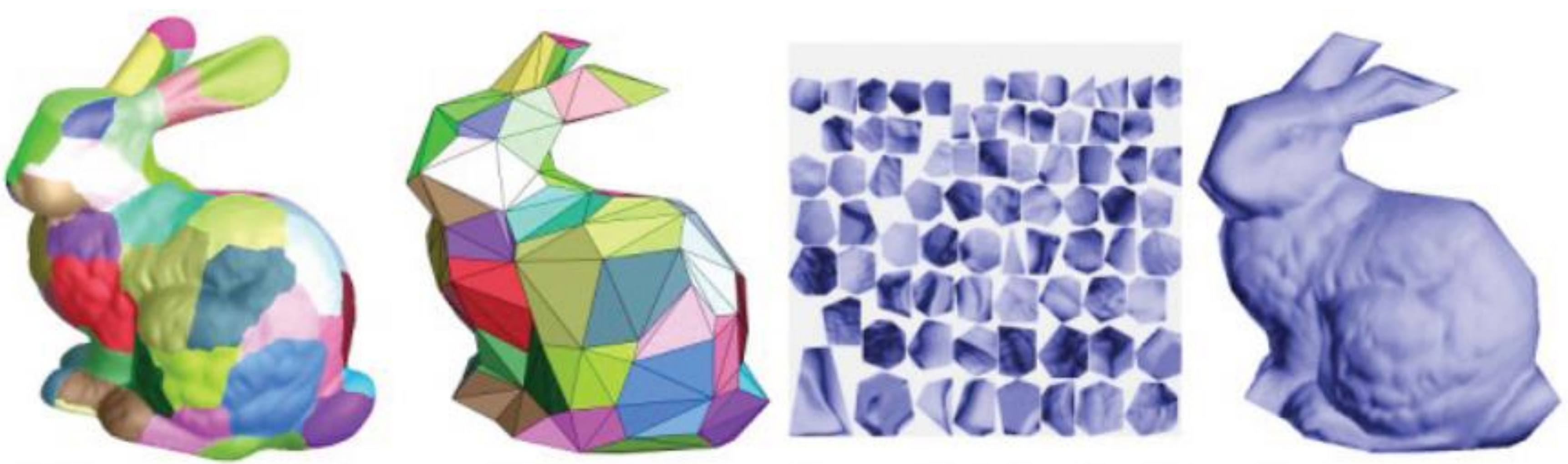
Parametrization in Practice

- Texture creation and parametrization is an art form
- Option: Unfold the surface



Parametrization in Practice

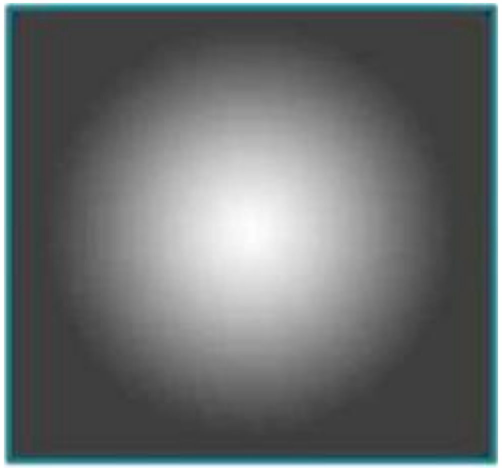
- Option: Create a Texture Atlas
- Break large mesh into smaller pieces



(a) charts on original mesh M (b) base mesh M (c) texture atlas (before pull-push) (d) textured base mesh

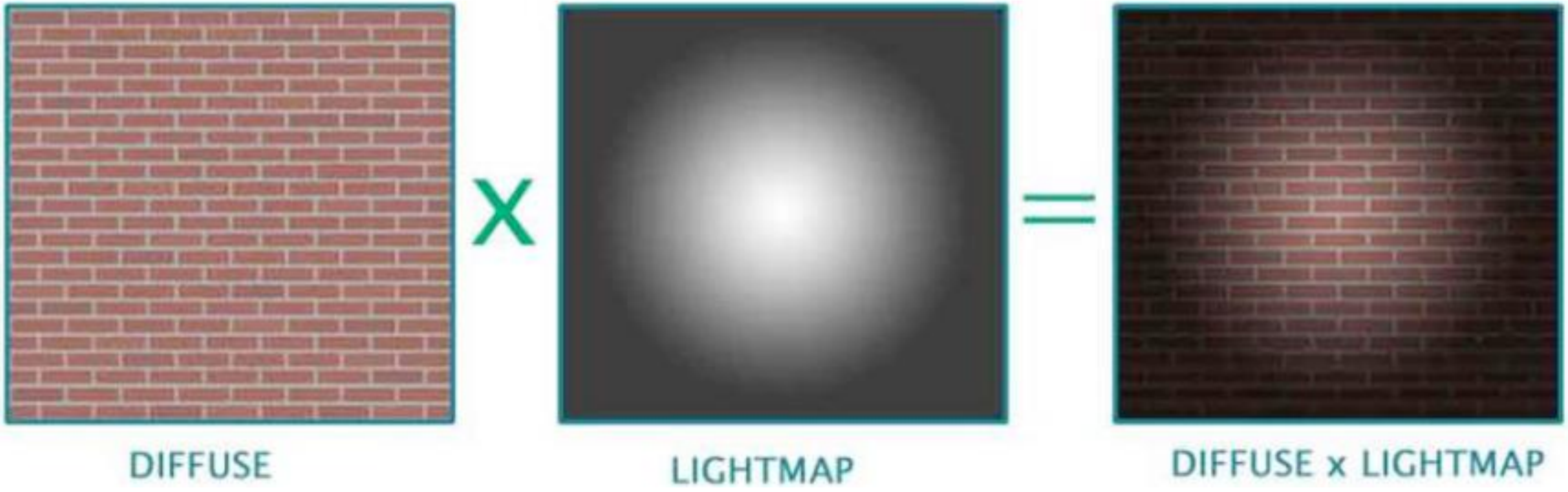
Light Maps

- Good shadows are complicated and expensive
- If lighting and objects will not change, neither are the shadows
- Can “bake” the shadows into a texture map as a preprocess step (called **lightmap**)
- During shading, lightmap values are multiplied into resulting pixel



LIGHTMAP

Light Maps



Specular Mapping

- Use a greyscale texture as a multiplier for the specular component

Alpha Mapping

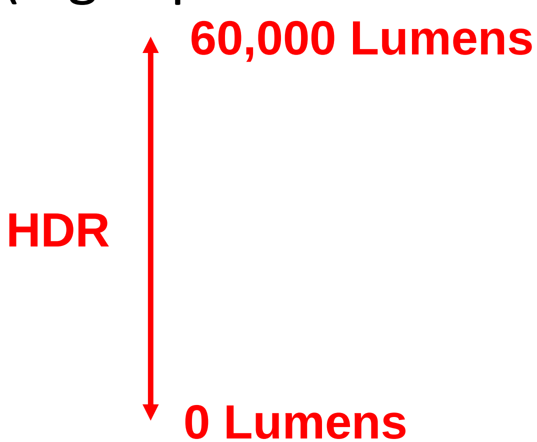
- Represent the texture in the alpha channel
- Can give complex outlines, used for plants

Render Bush
on 1 polygon

Render Bush
on polygon rotated
90 degrees

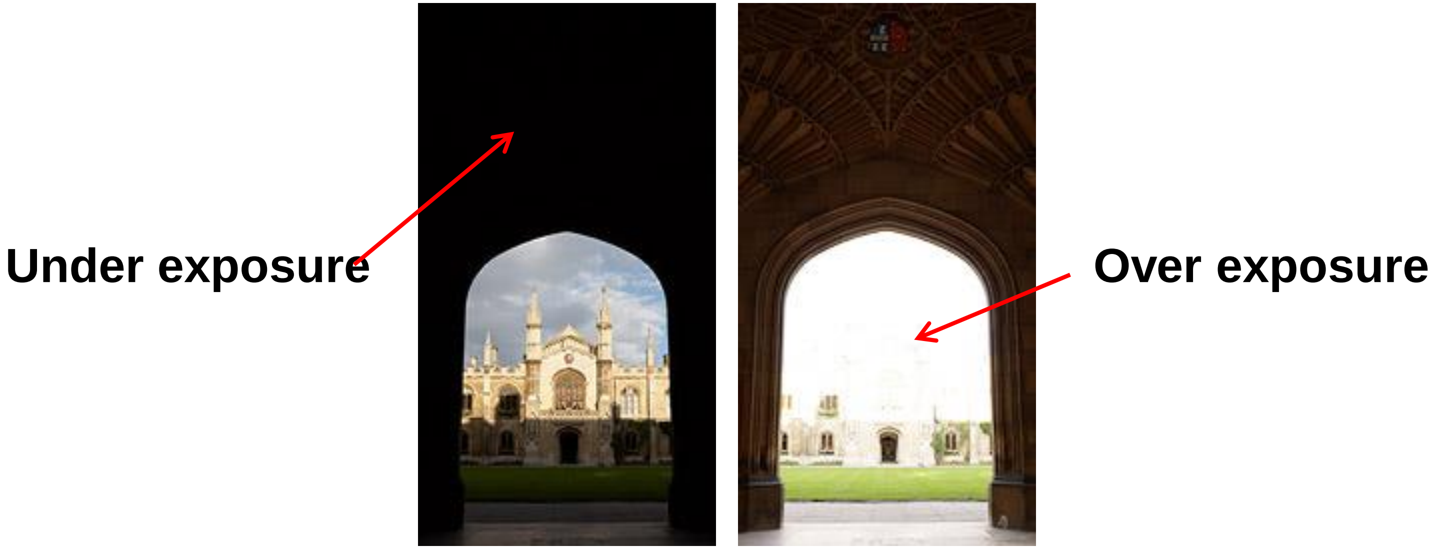
High Dynamic Range

- Sun's brightness is about 60,000 lumens
- Dark areas of earth has brightness of 0 lumens
- Basically, world around us has range of 0 – 60,000 lumens **(High Dynamic Range)**
- However, monitor has ranges of colors between 0 – 255 **(Low Dynamic Range)**
- New file formats have been created for HDR images (wider ranges). (E.g. OpenEXR file format)



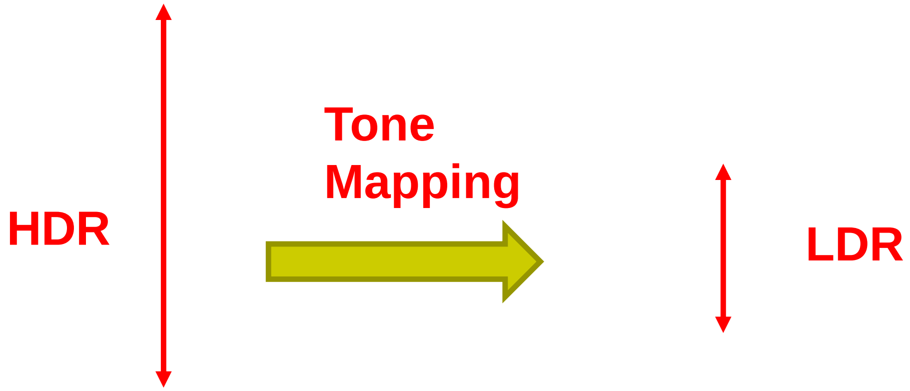
High Dynamic Range

- Some scenes contain **very bright** + **very dark** areas
- Using uniform scaling factor to map actual intensity to displayed pixel intensity means:
  - Either some areas are unexposed, or
  - Some areas of picture are overexposed



Tone Mapping

- Technique for scaling intensities in real world images (e.g HDR images) to fit in displayable range
- Try to capture feeling of real scene: **non-trivial**
- **Example:** If coming out of dark tunnel, lights should seem bright
- **General idea:** apply different scaling factors to diffferent parts of the image



Tone Mapping

***Figure 10**. Scene from Lost Coast at Varying Exposure Levels*

Types of Tone Mapping Operators

- **Global:** Use same scaling factor for all pixels
- **Local:** Use different scaling factor for different parts of image
- **Time-dependent:** Scaling factor changes over time
- **Time independent:** Scaling factor does NOT change over time
- Real-time rendering usually does **NOT** implement local operators due to their complexity

Simple (Global) Tone Mapping Methods

Maping to mean value

Division by maximum

Clipping on value 1

Interval mapping (interactive calibration)

Exponential mapping

Motion Blur

- Motion blur caused by exposing film to moving objects
- Motion blur: Blurring of samples taken over time (temporal)
- Makes fast moving scenes appear less jerky
- 30 fps + motion blur better than 60 fps + no motion blur

Motion Blur

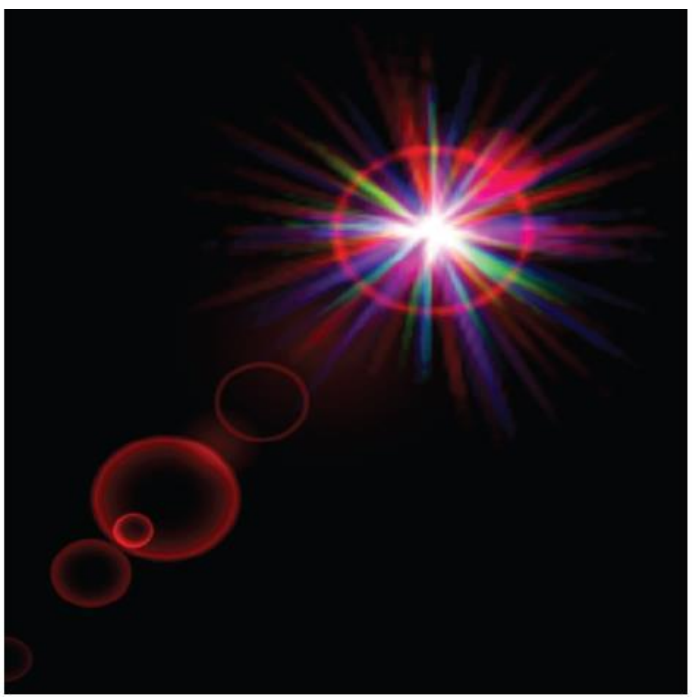
- Basic idea is to average series of images over time
- Move object to set of positions occupied in a frame, blend resulting images together
- Can blur moving average of frames. E.g blur 8 images
- **Velocity buffer:** blur in screen space using velocity of objects

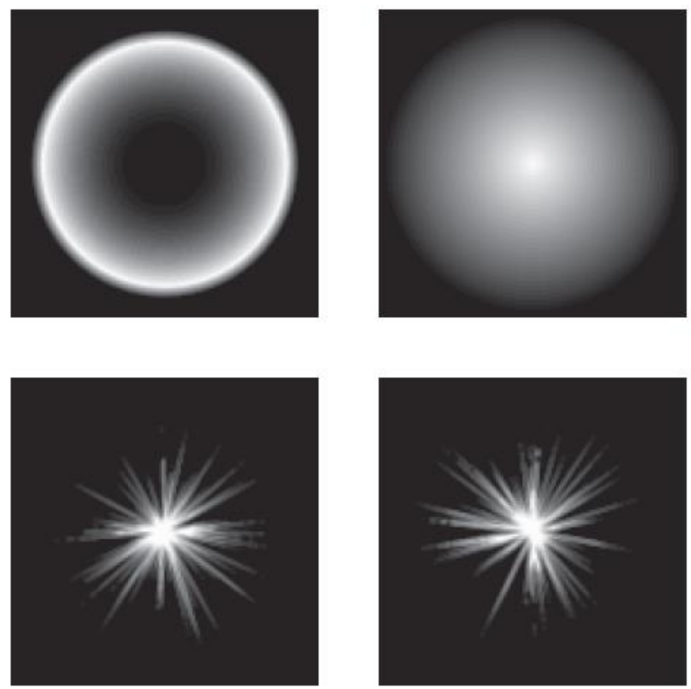
Depth of Field

- We can simulate a real camera
- In photographs, a range of pixels in focus
- Pixels outside this range are out of focus
- This effect is known as **Depth of field**

Lens Flare and Bloom

- Caused by lens of eye/camera when directed at light
- Halo – refraction of light by lens
- Ciliary Corona – Density fluctuations of lens
- Bloom – Scattering in lens, glow around light





Halo, Bloom, Ciliary Corona – top to bottom

Reference

- Tomas Akenine-Moller, Eric Haines and Naty Hoffman, Real Time Rendering