Computer Graphics (CS 543)
Lecture 10b: Soft Shadows (Maps and Volumes), Normal and Bump Mapping

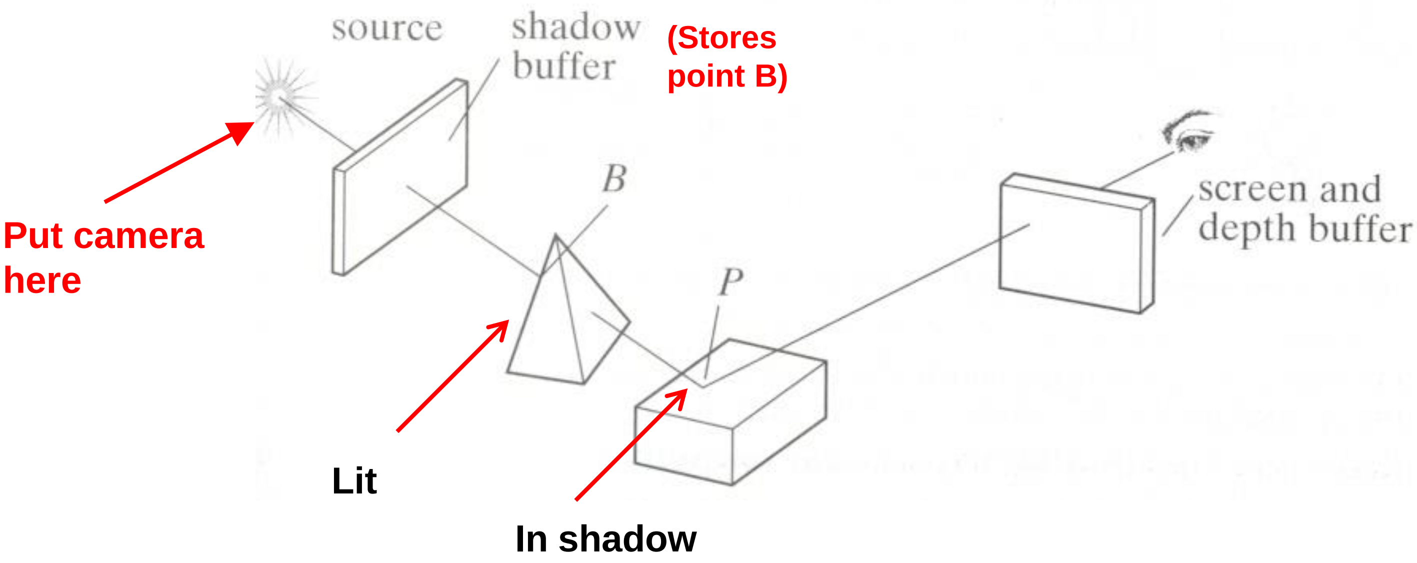
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

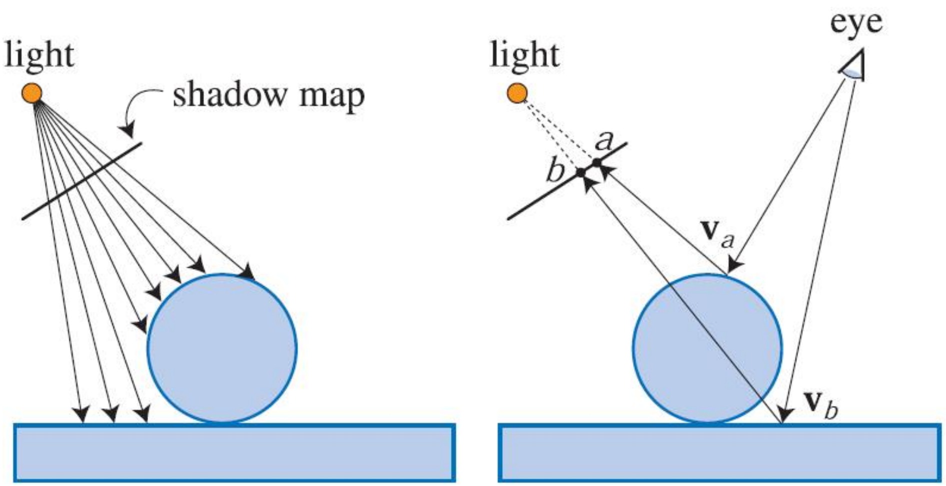
Shadow Buffer Theory

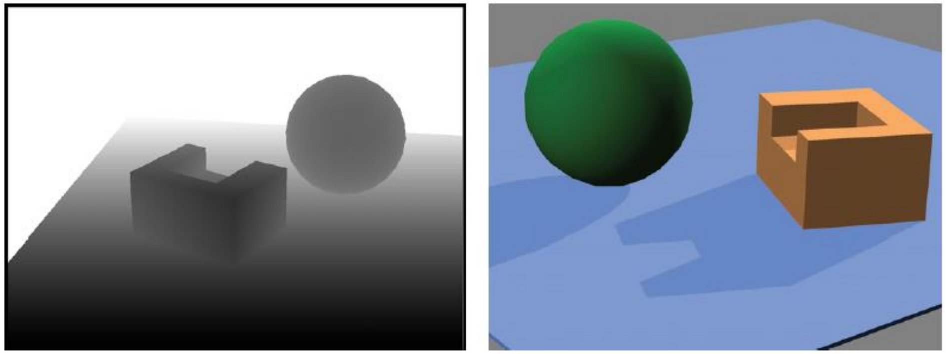
- **Observation:** Along each path from light
  - Only closest object is lit
  - Other objects on that path in shadow
- Shadow Buffer Method
  - Position a camera at light source.
  - uses second depth buffer called the **shadow map**
  - Shadow buffer stores closest object on each path



Shadow Map Illustrated

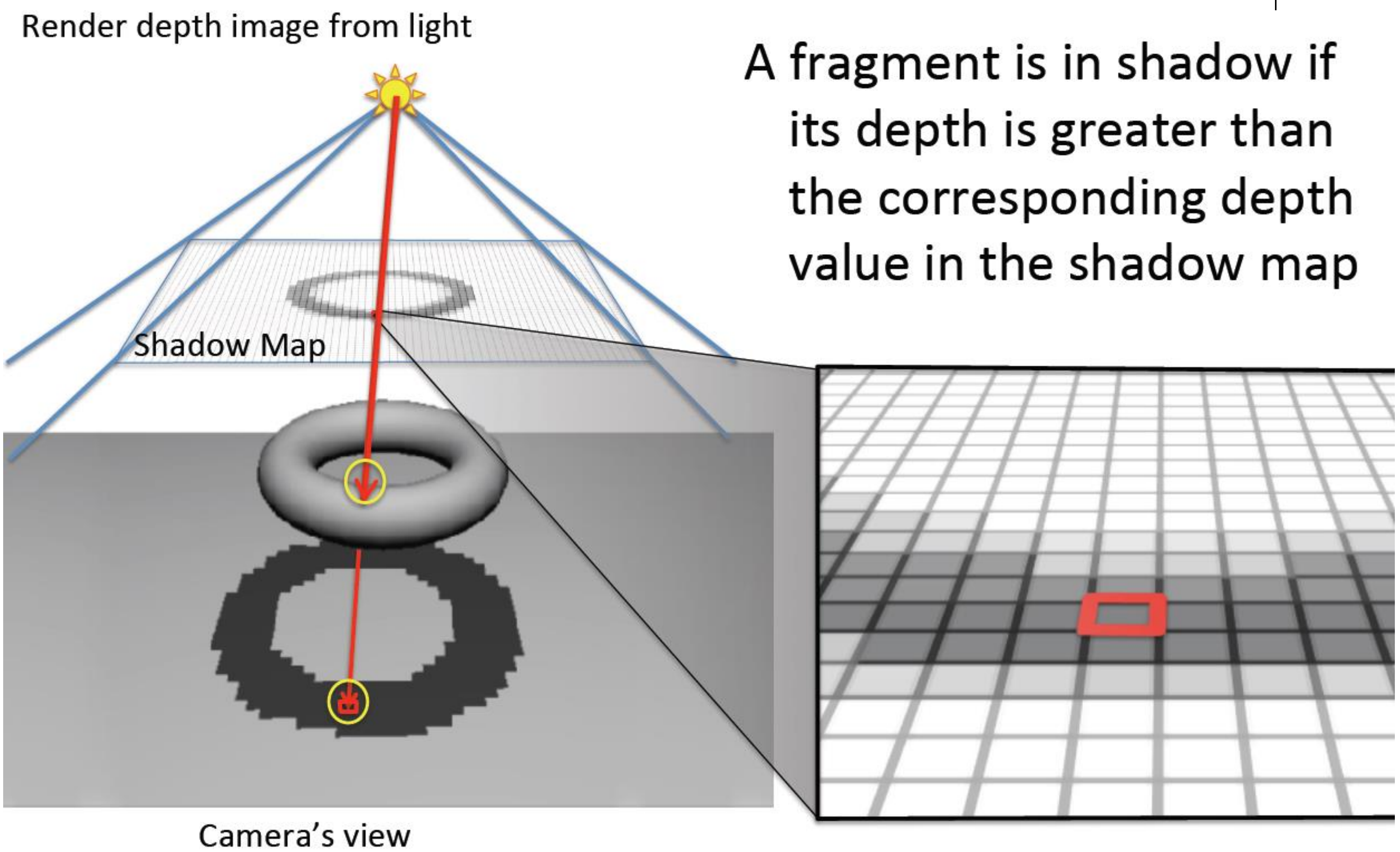
- Point v_a stored in element a of shadow map: lit!
- Point v_b **NOT** in element b of shadow map: In shadow





Not limited to planes

Shadow Map: Depth Comparison

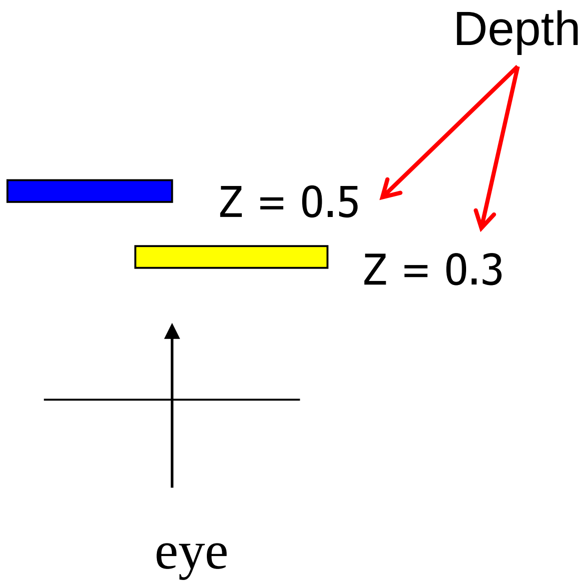


A fragment is in shadow if its depth is greater than the corresponding depth value in the shadow map

Recall: OpenGL Depth Buffer (Z Buffer)

- **Depth:** While drawing objects, depth buffer stores distance of each polygon from viewer
- **Why?** If multiple polygons overlap a pixel, only closest one polygon is drawn

1.0	1.0	1.0	1.0
1.0	0.3	0.3	1.0
0.5	0.3	0.3	1.0
0.5	0.5	1.0	1.0

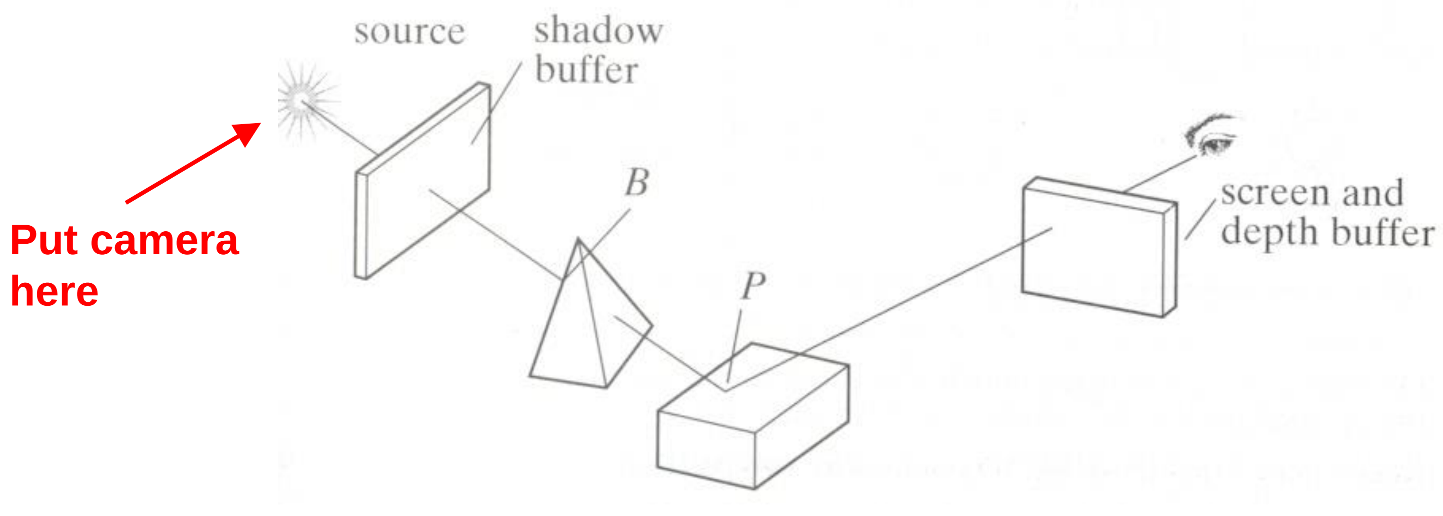


Shadow Map Approach

- Rendering in two stages:
  - Generate/load shadow Map
  - Render the scene

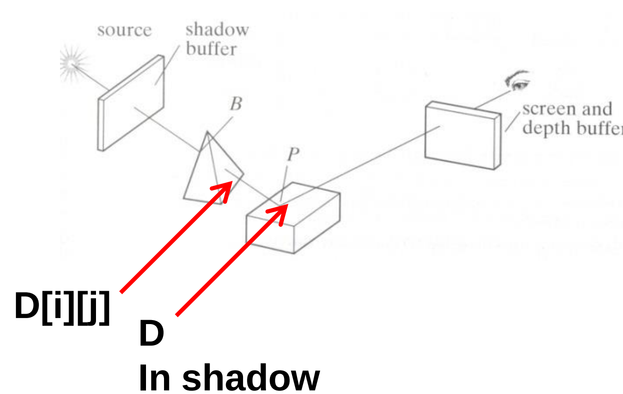
Loading Shadow Map

- Initialize each element to 1.0
- Position a camera at light source
- Rasterize each face in scene updating closest object
- Shadow map (buffer) tracks smallest depth on each path



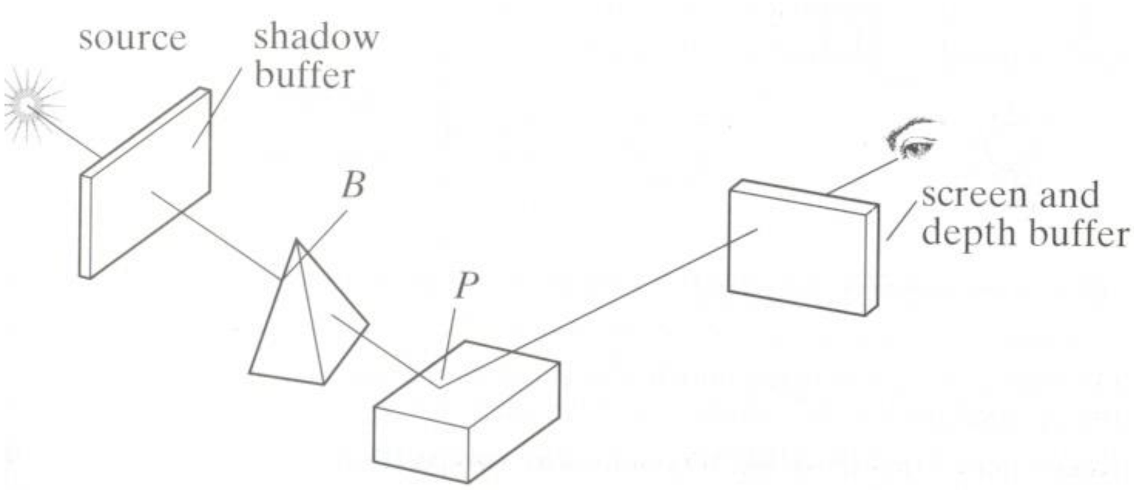
Shadow Map (Rendering Scene)

- Render scene using camera as usual
- While rendering a pixel find:
  - pseudo-depth D from light source to P
  - Index location [i][j] in shadow buffer, to be tested
  - Value d[i][j] stored in shadow buffer
- If d[i][j] < D (other object on this path closer to light)
  - point P is in shadow
  - lighting = ambient
- Otherwise, not in shadow
  - Lighting = amb + diffuse + specular

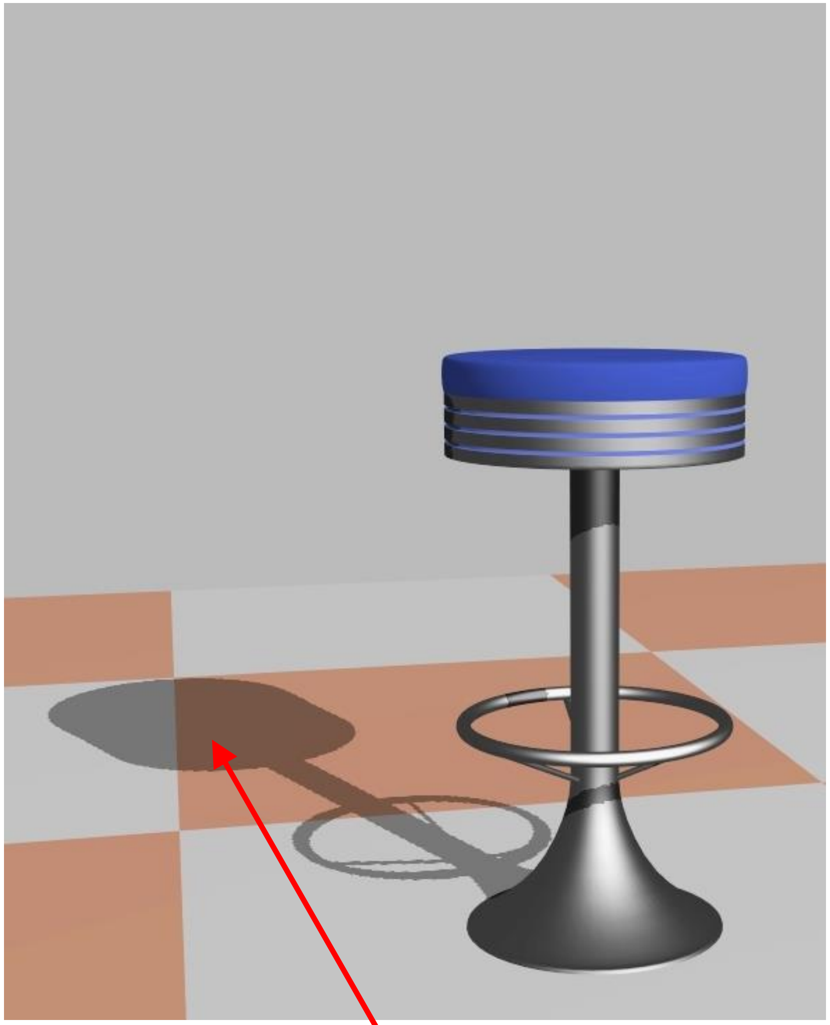


Loading Shadow Map

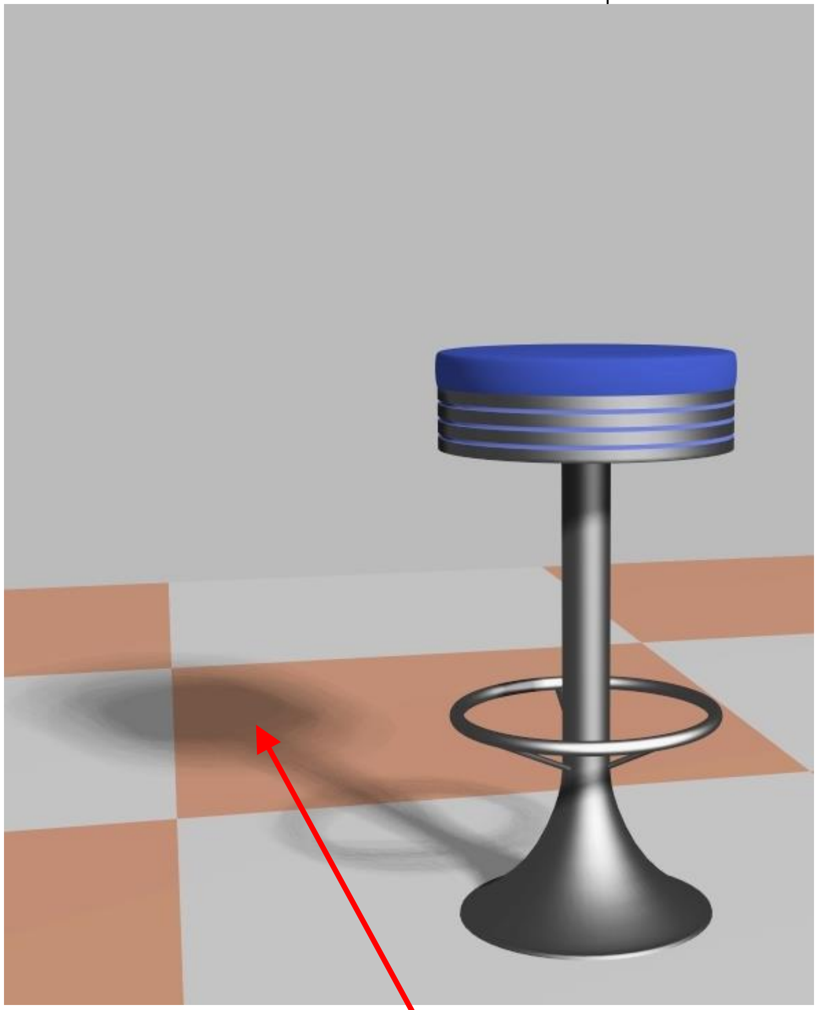
- Shadow map calculation is independent of eye position
- In animations, shadow map loaded once
- If eye moves, no need for recalculation
- If objects move, recalculation required



Example: Hard vs Soft Shadows



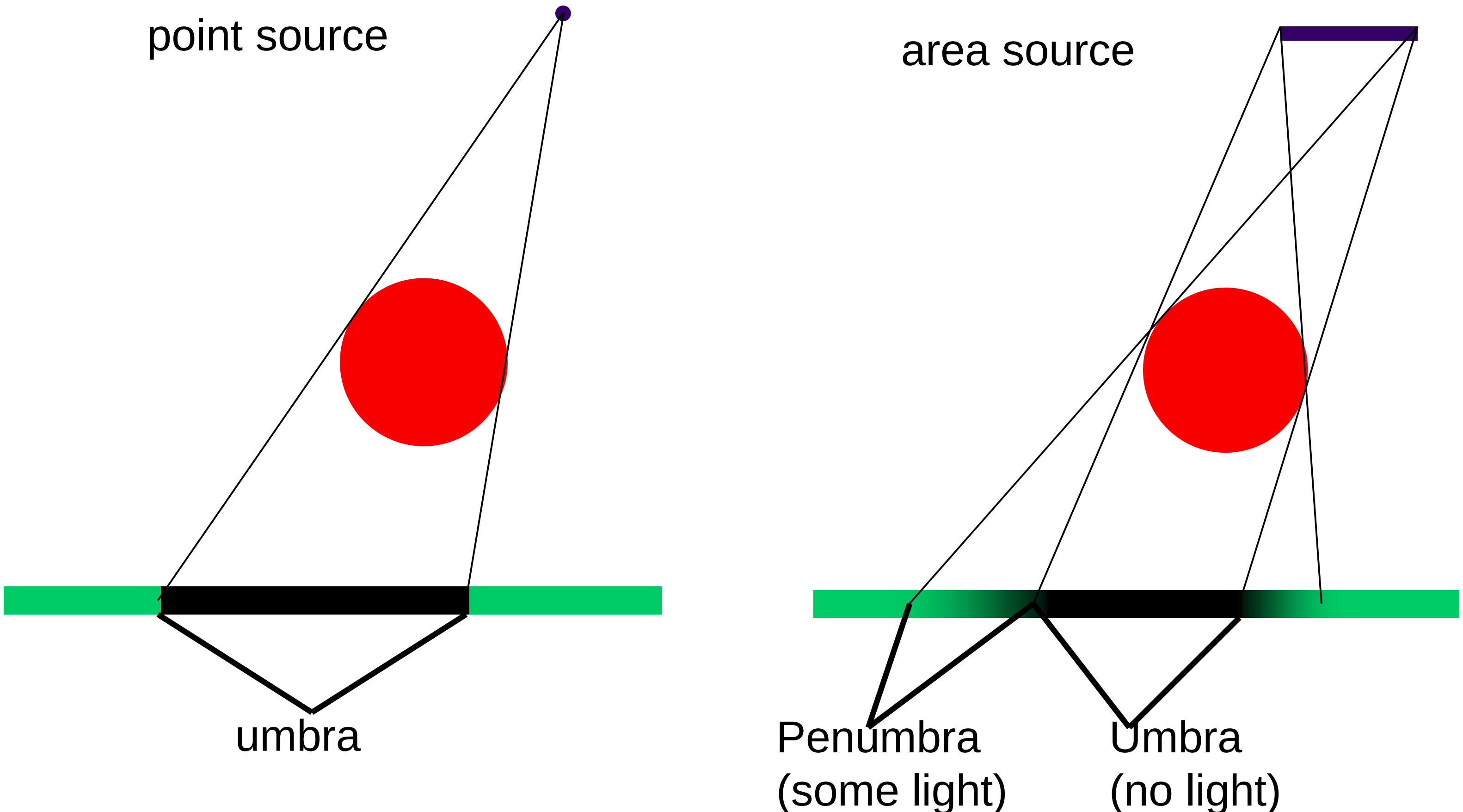
Hard Shadow



Soft Shadow

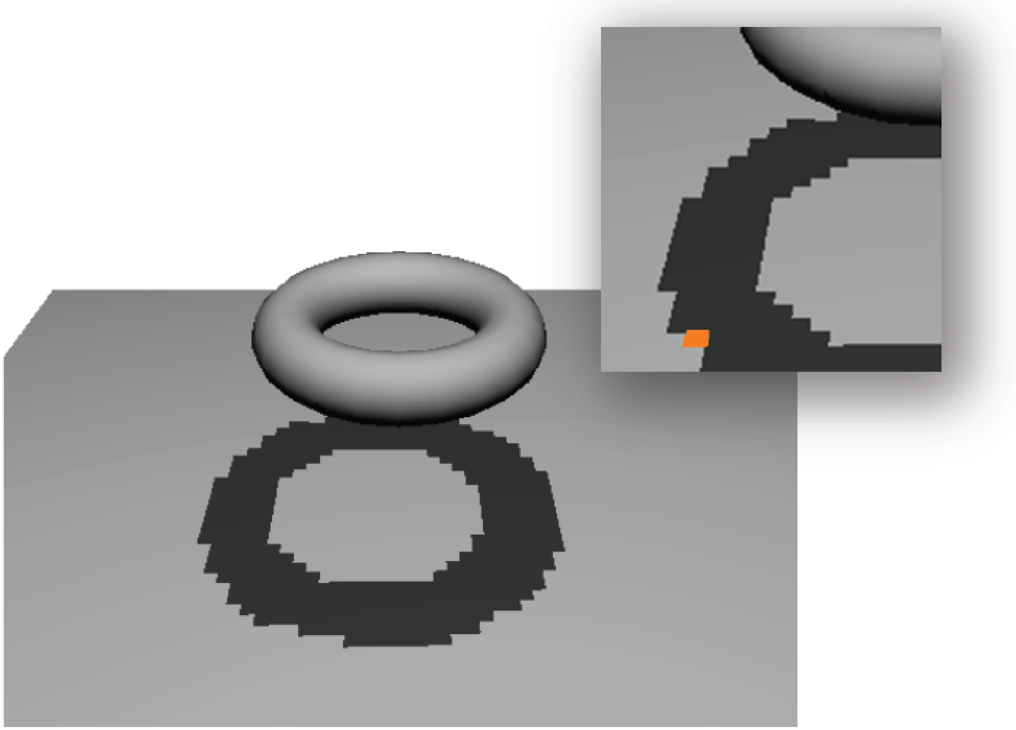
Definitions

- Point light: create hard shadows (unrealistic)
- Area light: create soft shadows (more realistic)

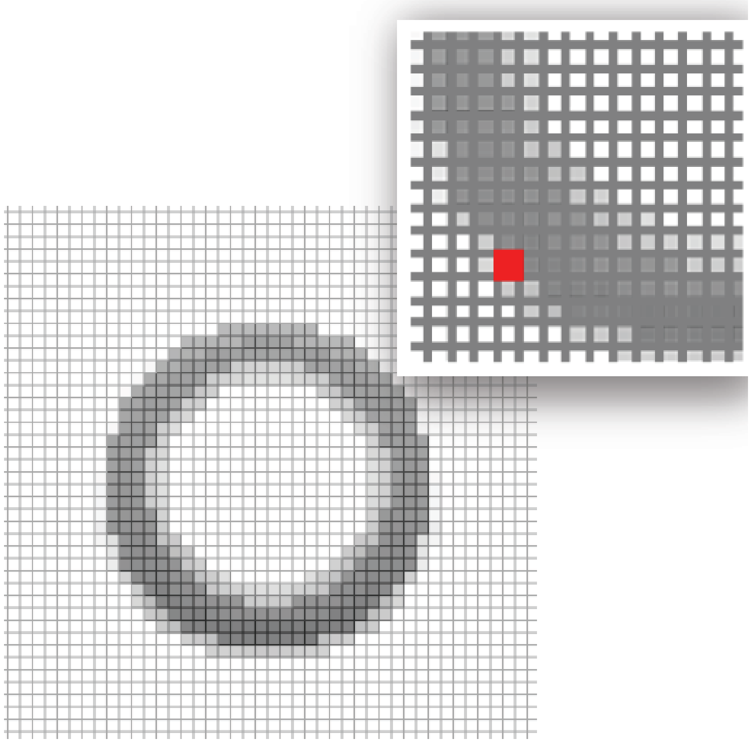


Shadow Map Problems

- Low shadow map resolution results in jagged shadows



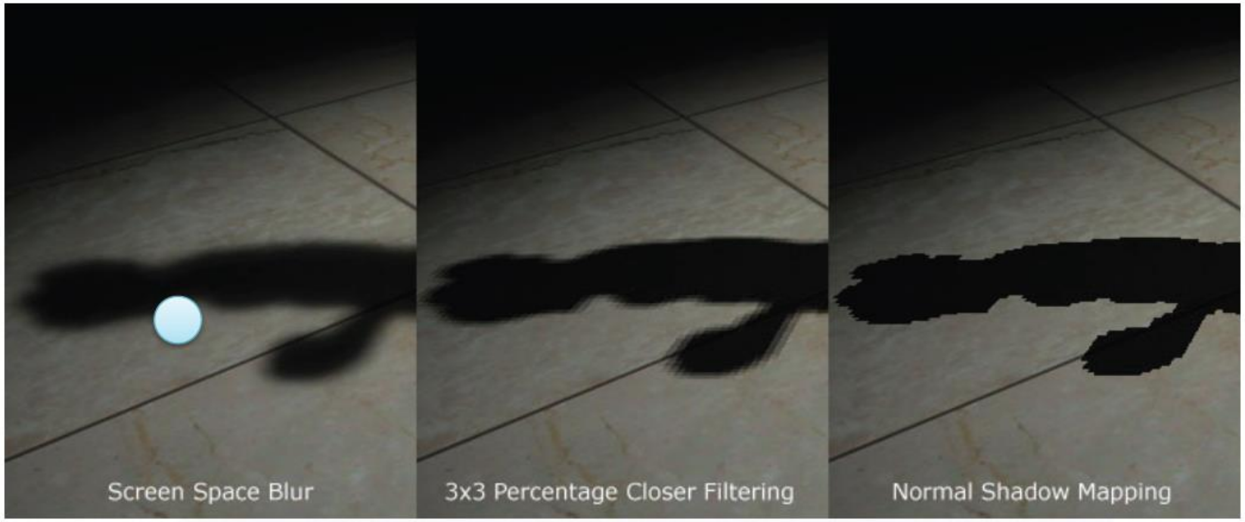
from viewpoint

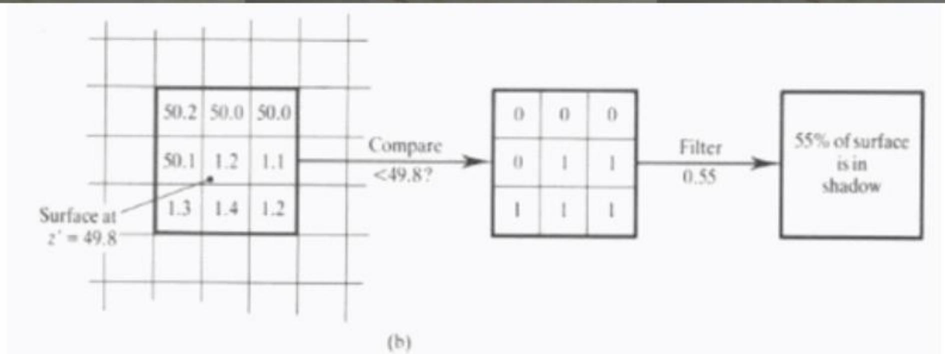


from light

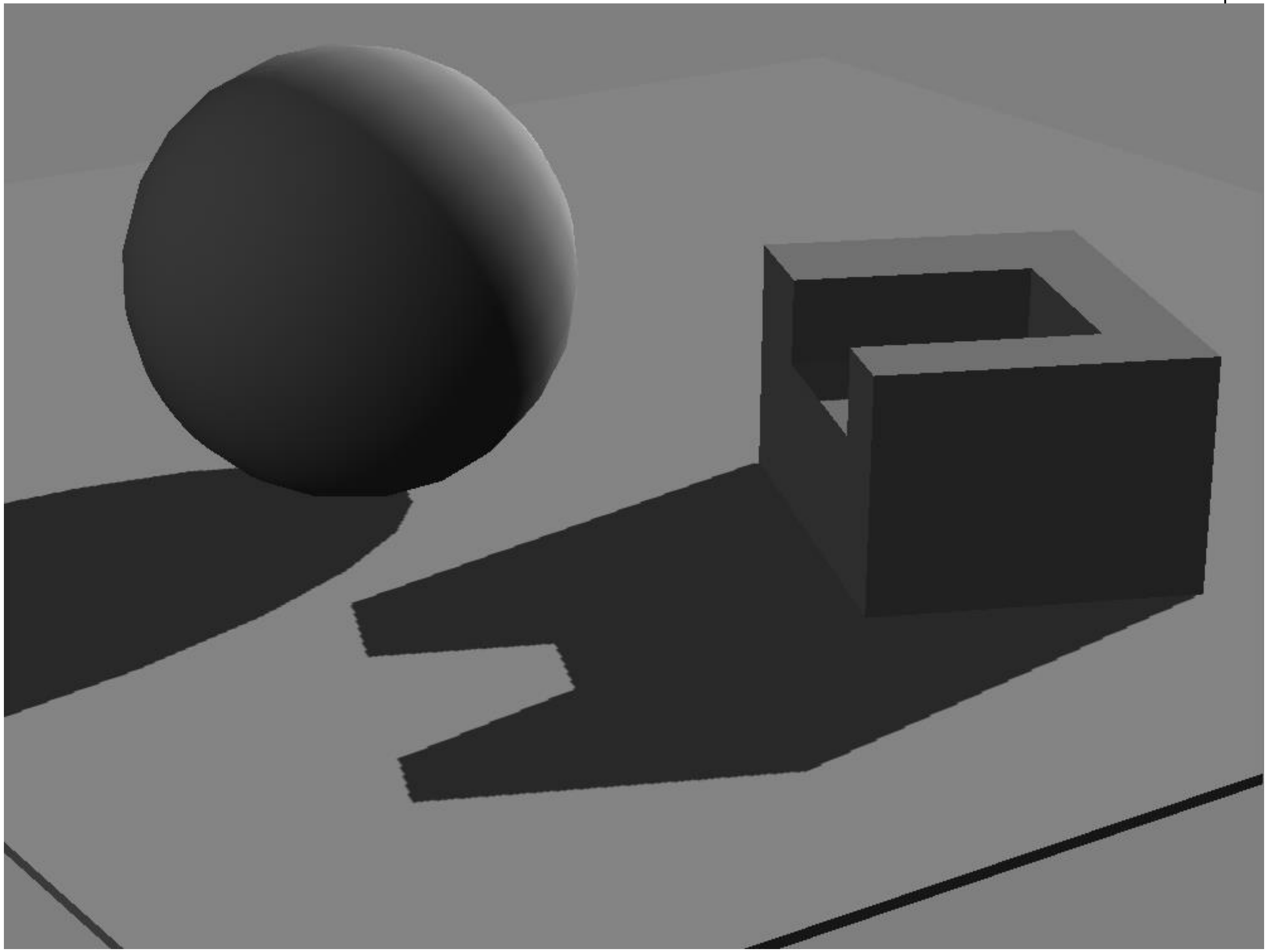
Percentage Closer Filtering

- Instead of retrieving just 1 value from shadow map, retrieve neighboring shadow map values as well
- Blend multiple shadow map samples to reduce jaggies



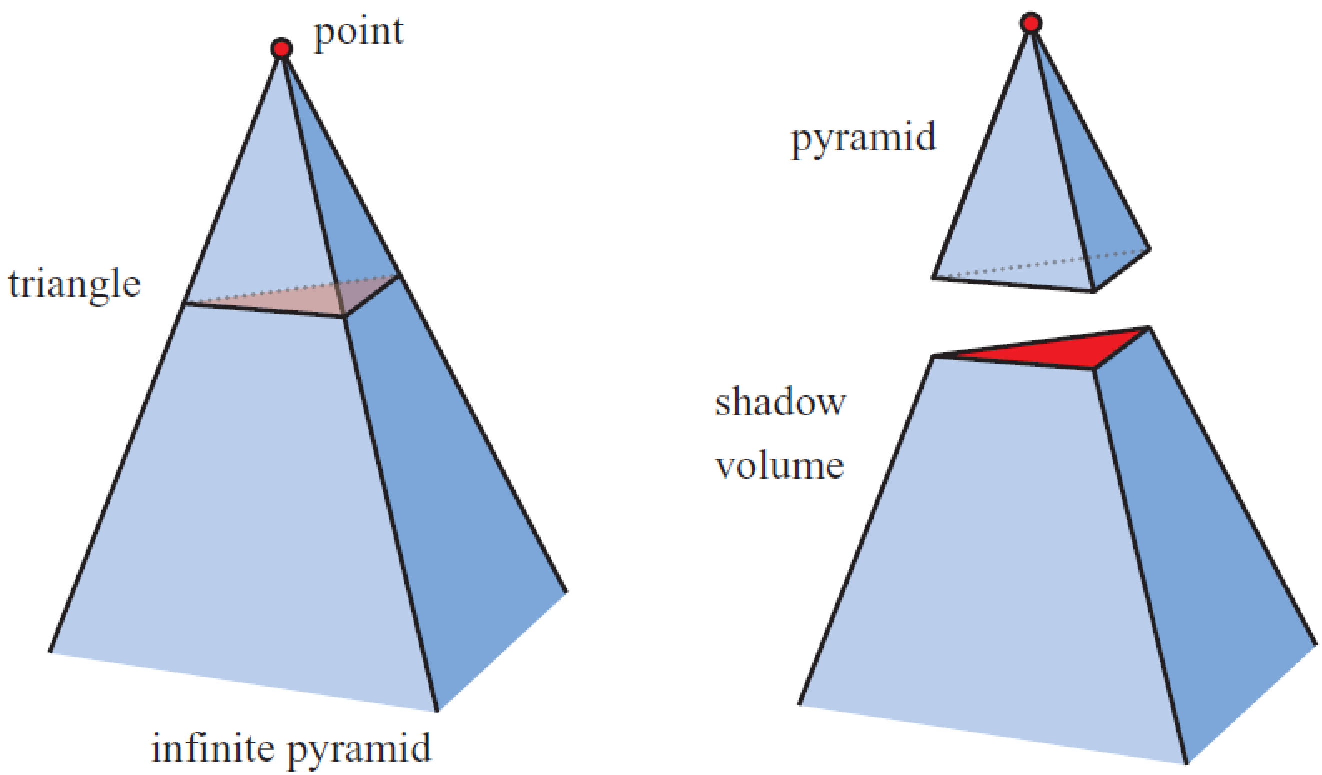


Shadow Map Result



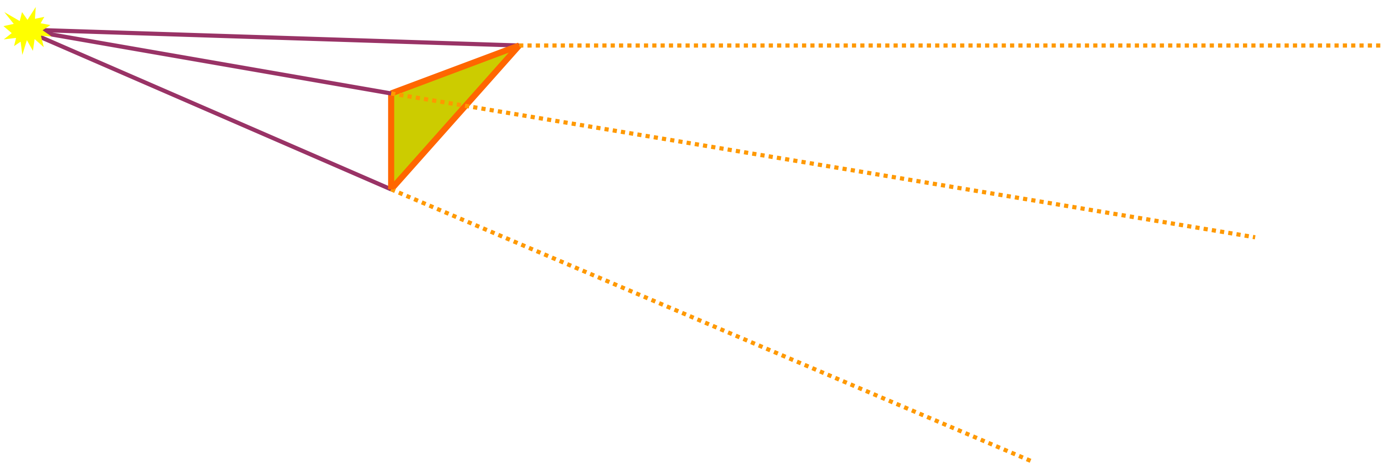
Shadow volumes

- Most popular method for real time
- Shadow volume concept



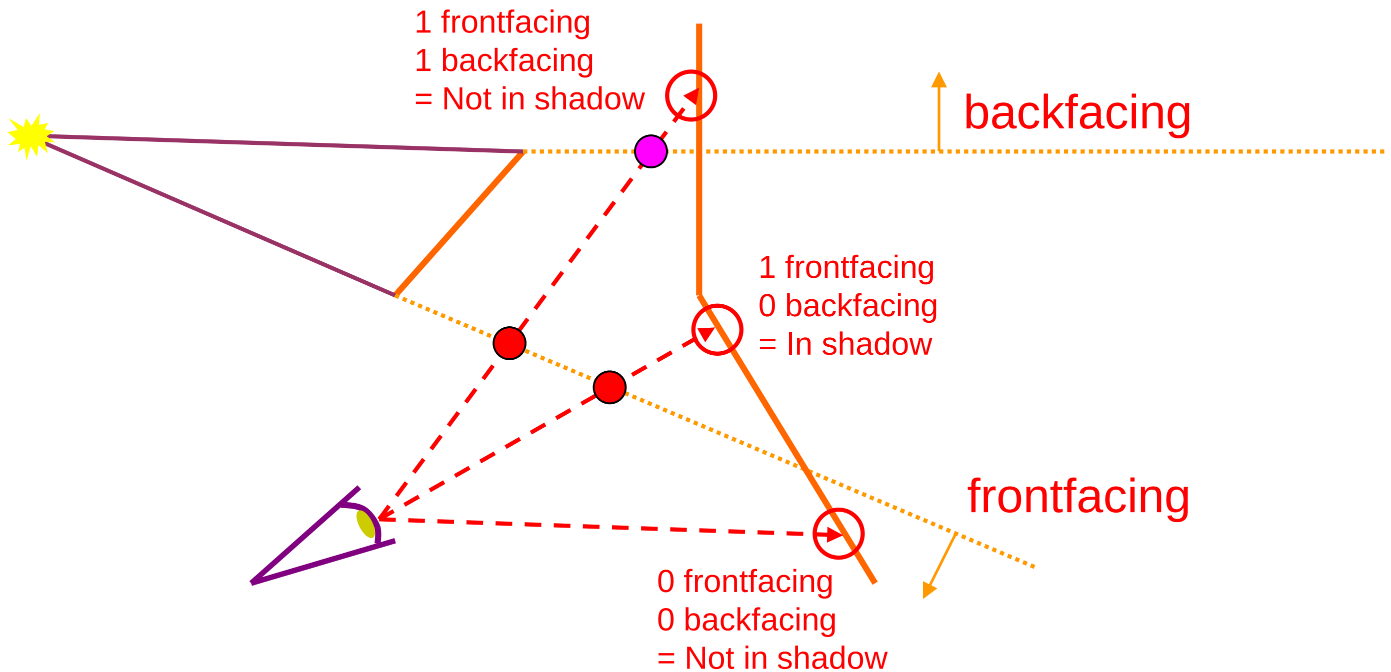
Shadow volumes

- Create volumes of space in shadow from each polygon in light
- Each triangle creates 3 projecting quads



Using Shadow Volume

- To test a point, count number of polygon intersections between the point and the eye.
- If we look through more frontfacing than backfacing polygons, then in shadow.



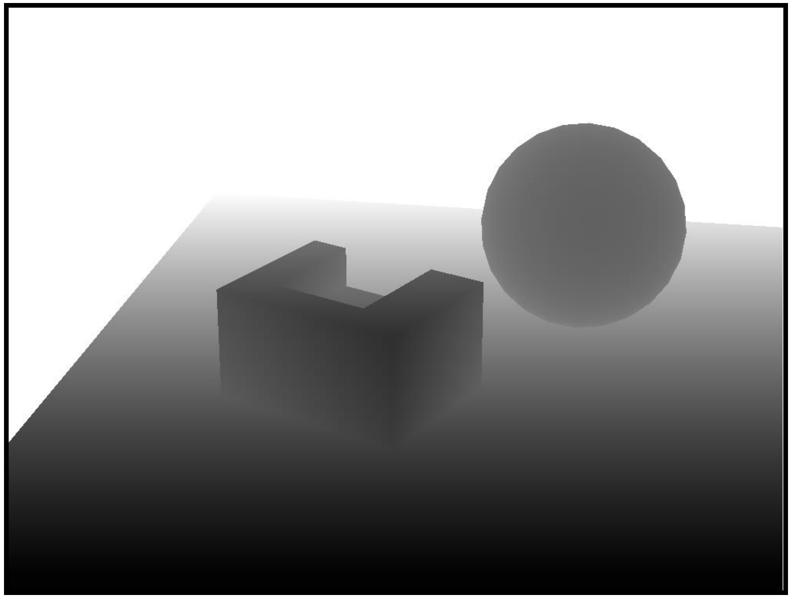
Shadow Volume Example

Image courtesy of NVIDIA Inc.

Arbitrary geometry

- Shadow mapping and shadow volumes can render shadows onto arbitrary geometry
  - Recent focus on shadow volumes, because currently most popular, and works on most hardware
- Works in real time…
- Shadow mapping is used

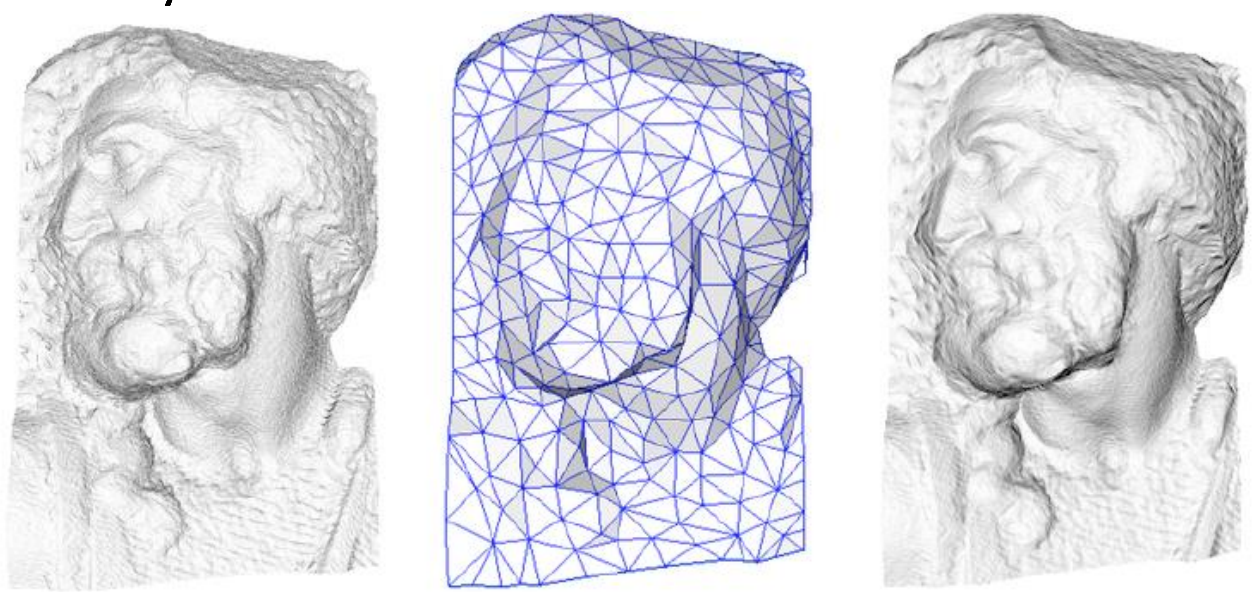
in Pixar’s rendering software



Normal Mapping

Normal Mapping

- Store normals in texture
- Normals <x,y,z> stored in <r,g,b> values in texture
- **Idea:** Use low resolution mesh + high resolution normal map
- 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

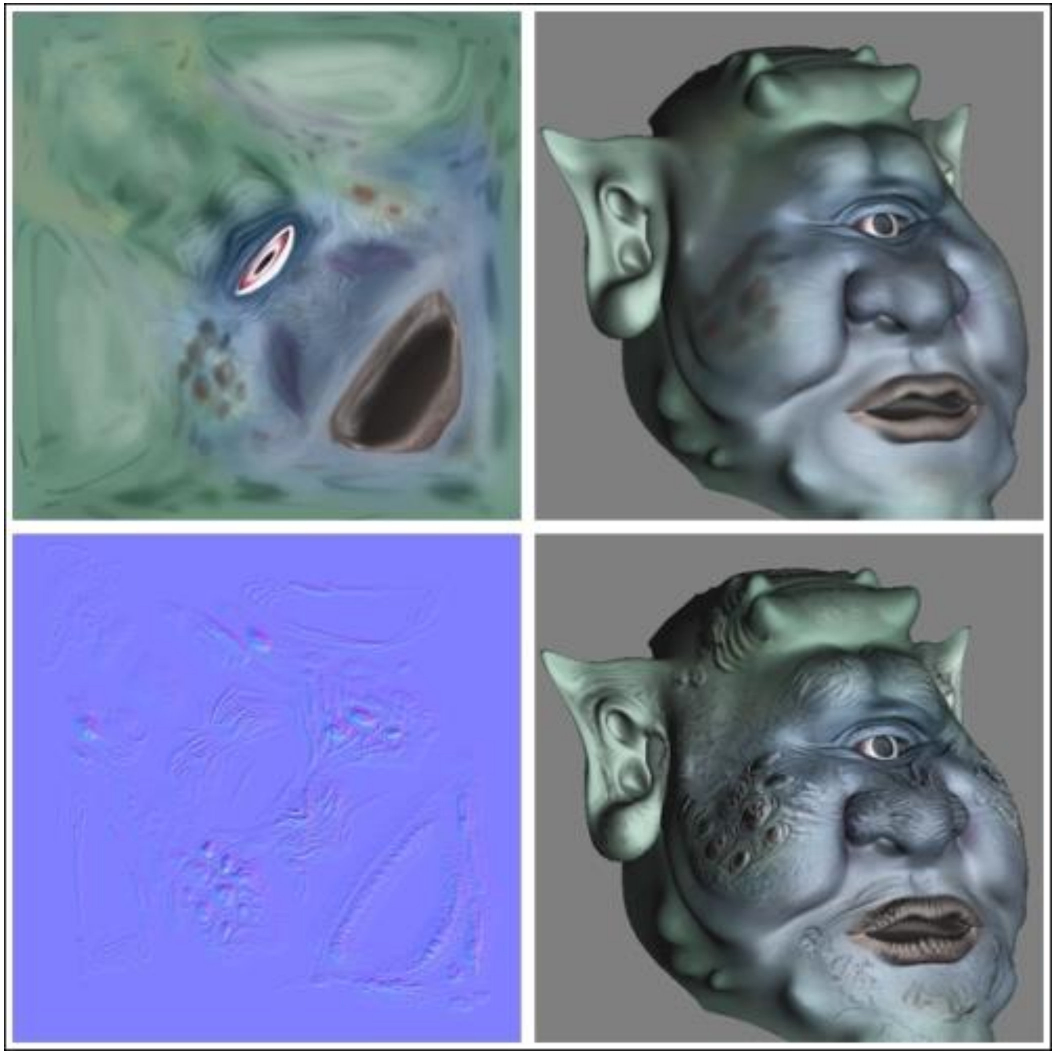
simplified mesh and normal mapping 500 triangles

Normal Mapping Example: Ogre

OpenGL 4 Shading Language Cookbook (3rd edition) by David Wolff (pg 157)

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

Normal texture map

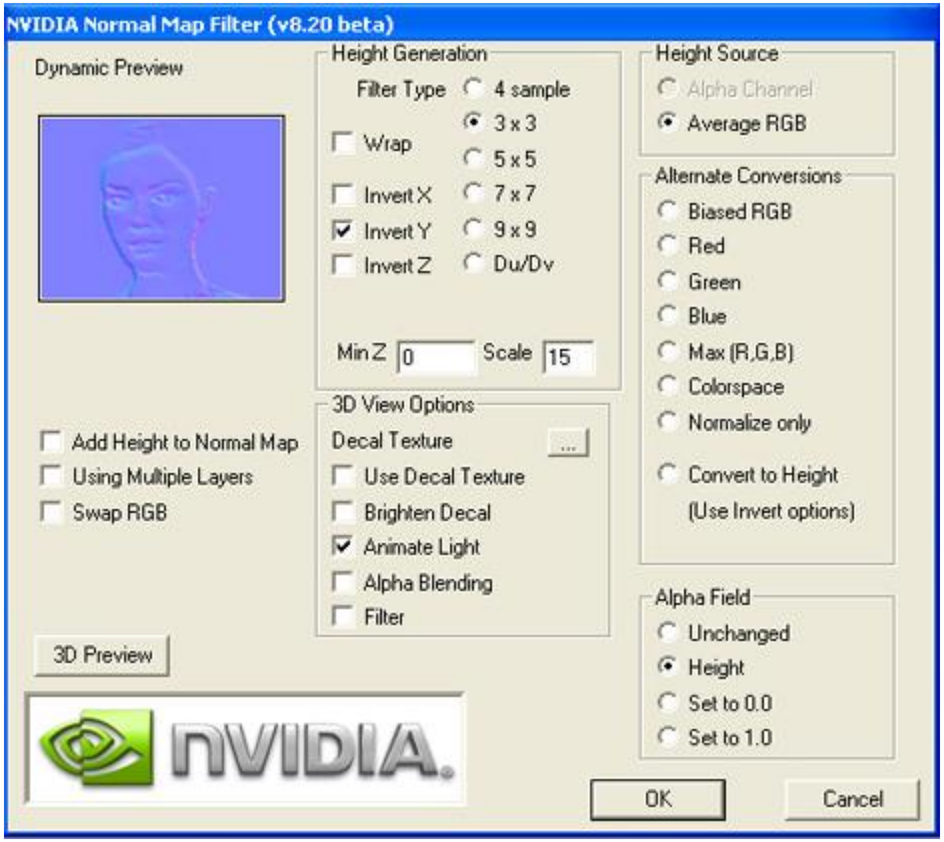


Texture mapped Ogre (Uses mesh normals)

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

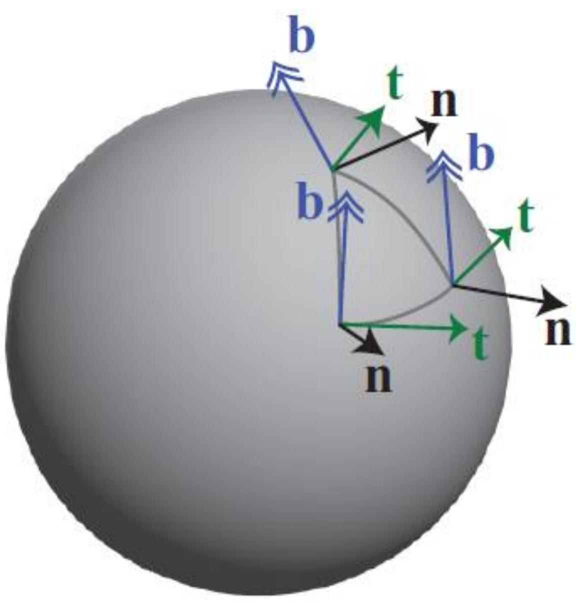
Creating Normal Maps

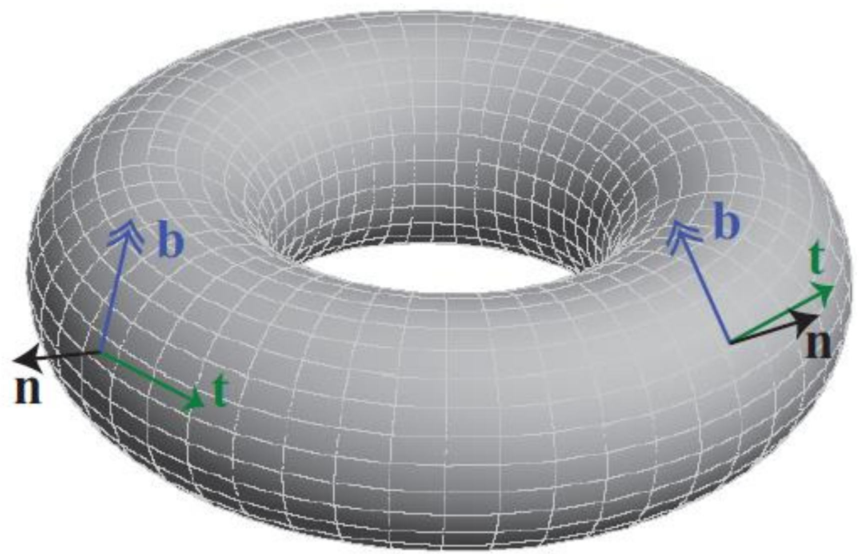
- Many tools for creating normal map
- E.g. Nvidia texture tools for Adobe photoshop
  - https://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop



Tangent Space Vectors

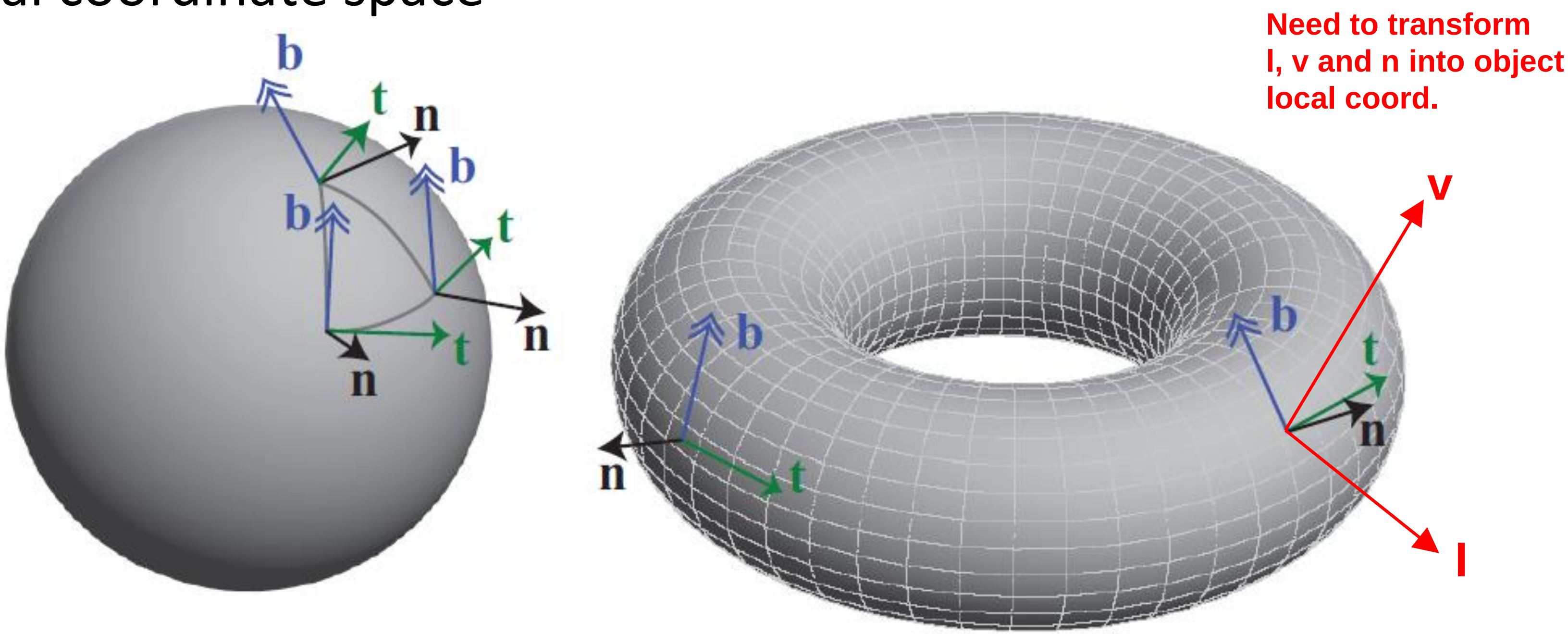
- Normals in normal map stored in object local coord. frame (or tangent space)
- Object Local coordinate space? Axis positioned on surface of object (NOT global x,y,z)
- Need Tangent, normal and bi-tangent vectors at each vertex
  - z axis aligned with mesh normal at that point
  -



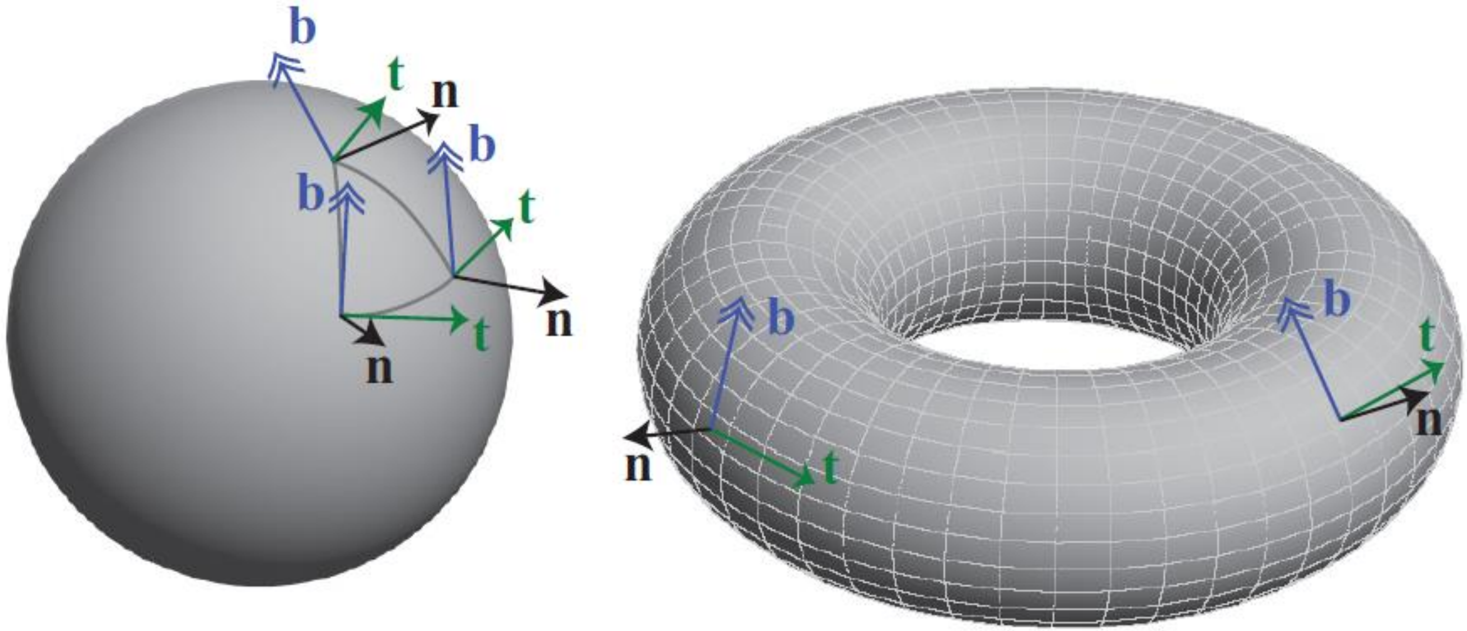


Tangent Space Vectors

- 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



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



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

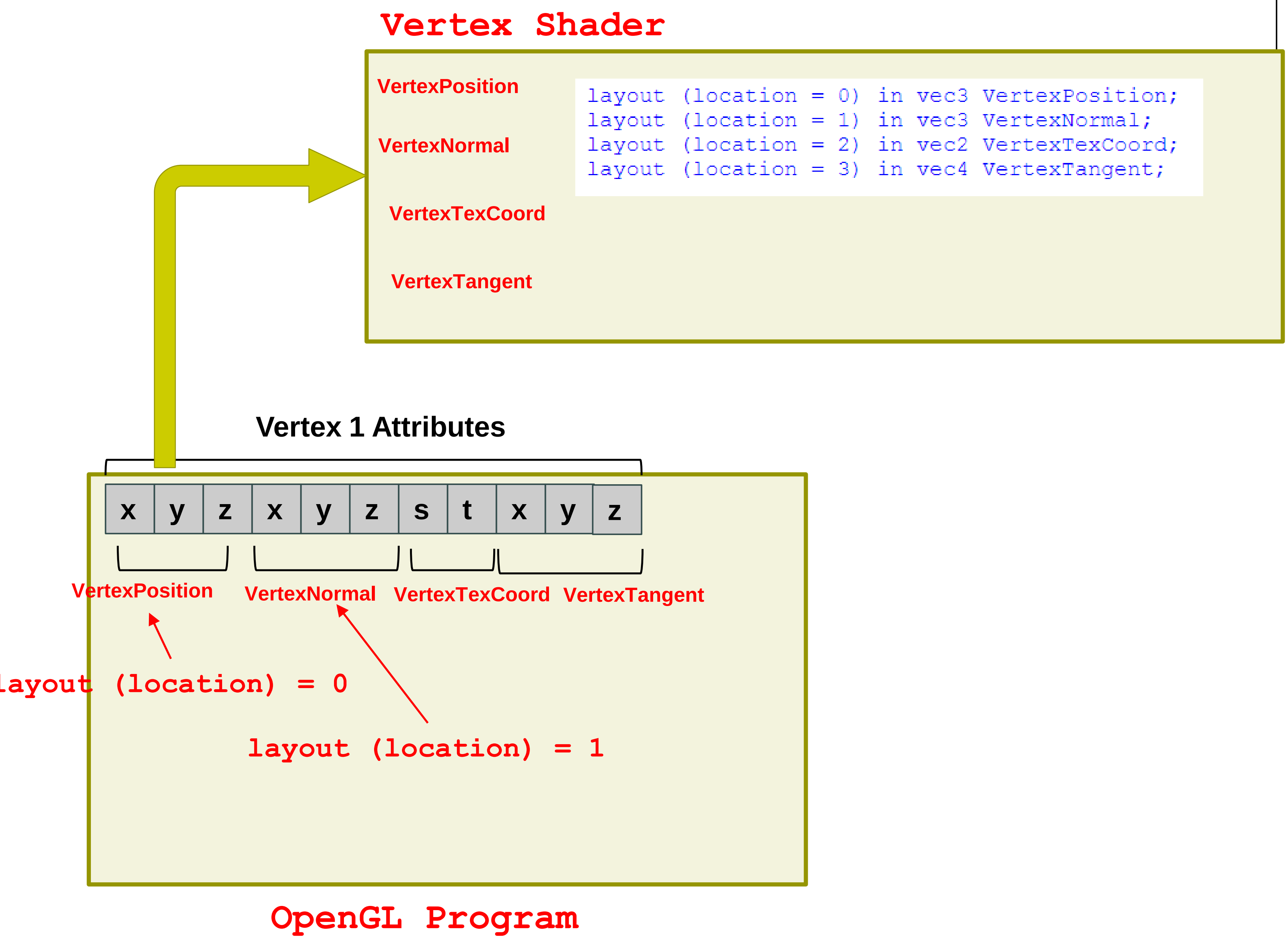
Point *S* in object's local coordinate frame

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

Point *P* in eye coordinate frame

Normal Mapping Example

OpenGL 4 Shading Language Cookbook (3rd edition) by David Wolff (pg 159)



Normal Mapping Example

OpenGL 4 Shading Language Cookbook (3rd edition) by David Wolff (pg 159)

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;
```

....

```
// 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 ) ;
```

Transform normal and tangent to eye space
….
Compute bi-normal vector

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 (3rd edition) by David Wolff (pg 159)

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;
```

……………………

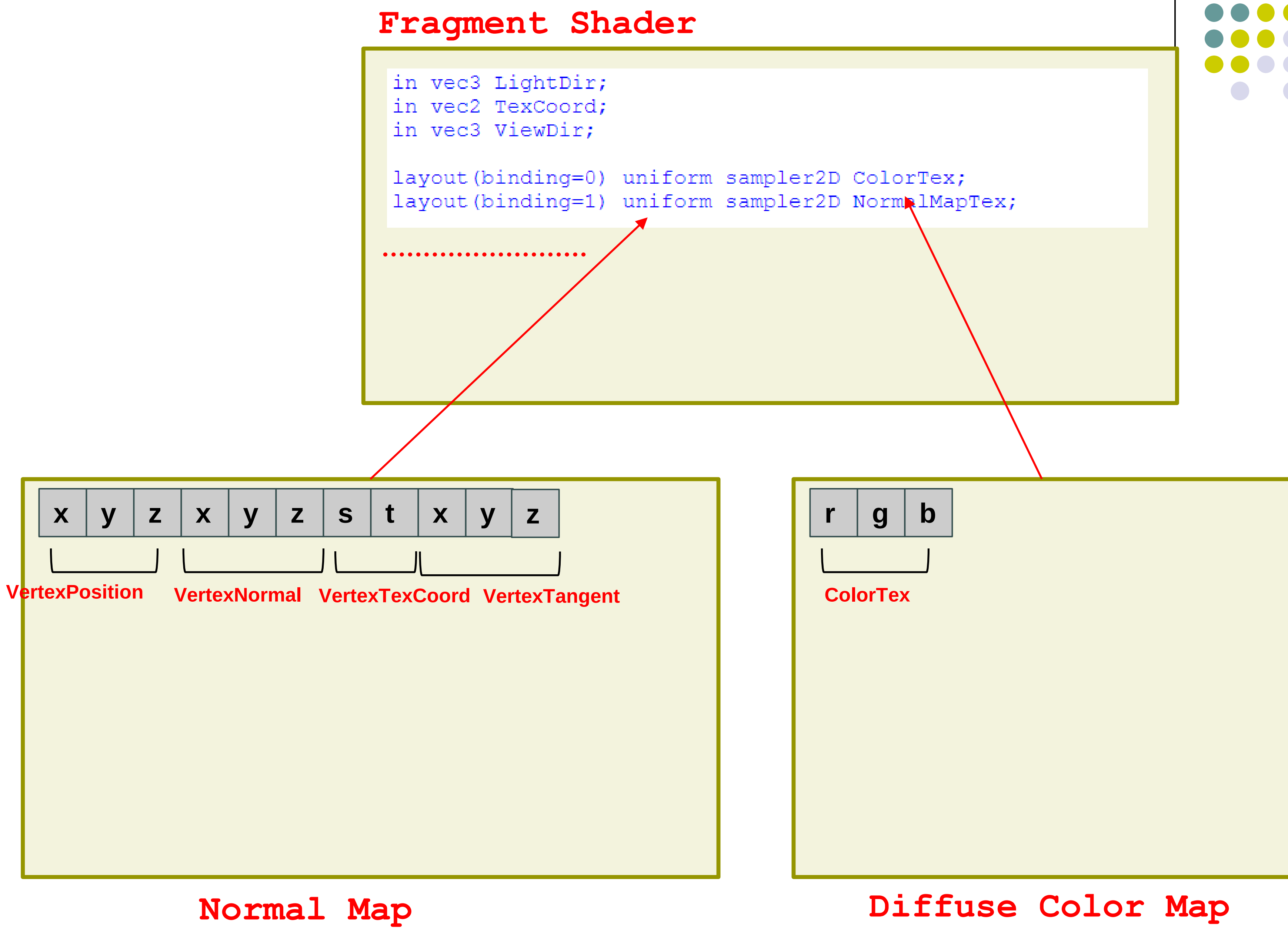
Receive Light, View directions and TexCoord set in vertex shader

……

Declare Normal and Color maps

Normal Mapping Example

OpenGL 4 Shading Language Cookbook (3rd edition) by David Wolff (pg 159)



Normal Mapping Example

OpenGL 4 Shading Language Cookbook (3rd edition) by David Wolff (pg 159)

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
Rescale from [0,1] to [-1,1] range

Look up diffuse coeff. from color texture

x	y	z	x	y	z	s	t	x	y	z

VertexPosition VertexNormal VertexTexCoord VertexTangent

Normal Map

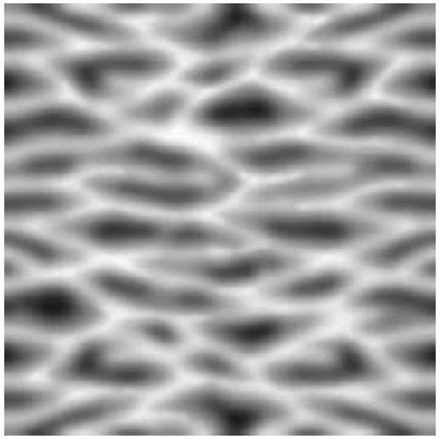
r	g	b

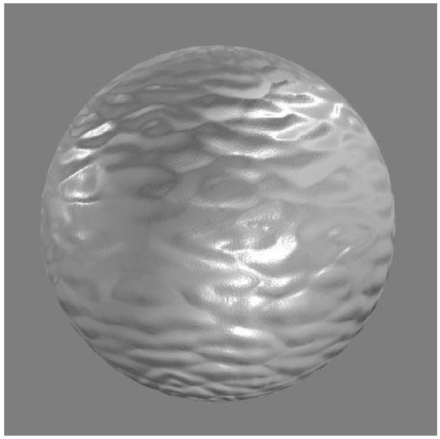
ColorTex

Diffuse Color Map

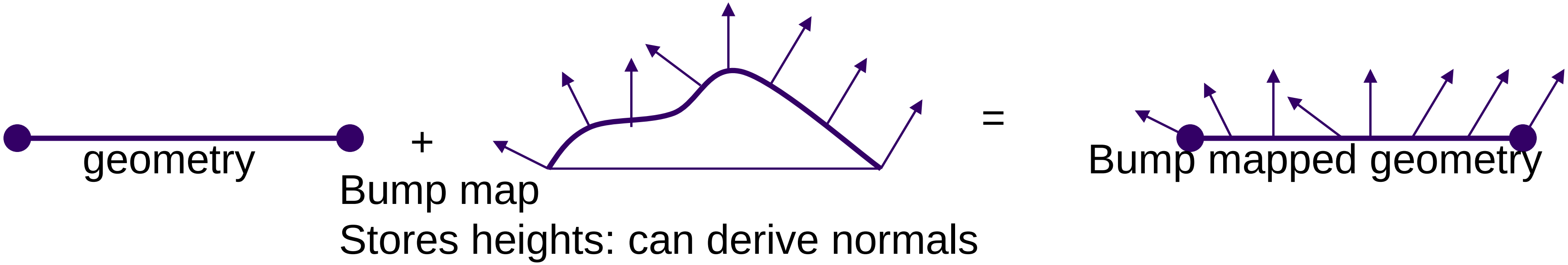
Bump Mapping

Bump mapping





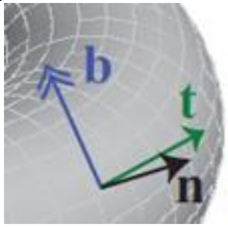
- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
  - Too expensive to model these geometrically
- Instead let a texture modify the normal at each pixel, and then use this normal to compute lighting

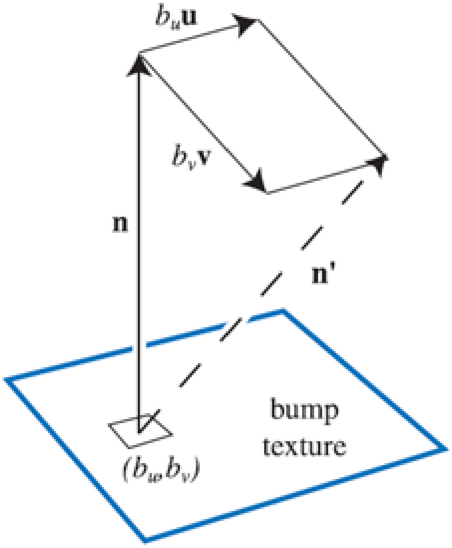


Bump mapping: Blinn's method

- **Idea:** Distort the surface normal at point to be rendered
- **Option a (left):** Modify normal **n** along u, v axes to give **n'**
  - In texture map, store how much to perturb **n** (**b_u** and **b_v**)
- Using bumpmap
  - Look up **b_u** and **b_v**
  - **$n' = n + b_u T + b_v B$**

(T and B are tangent and bi-tangent vectors)

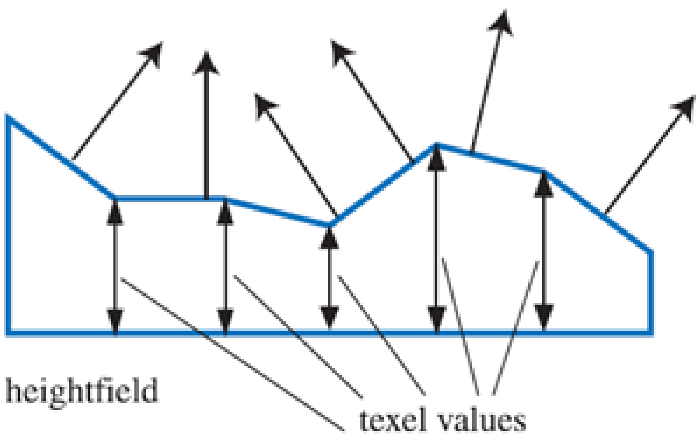




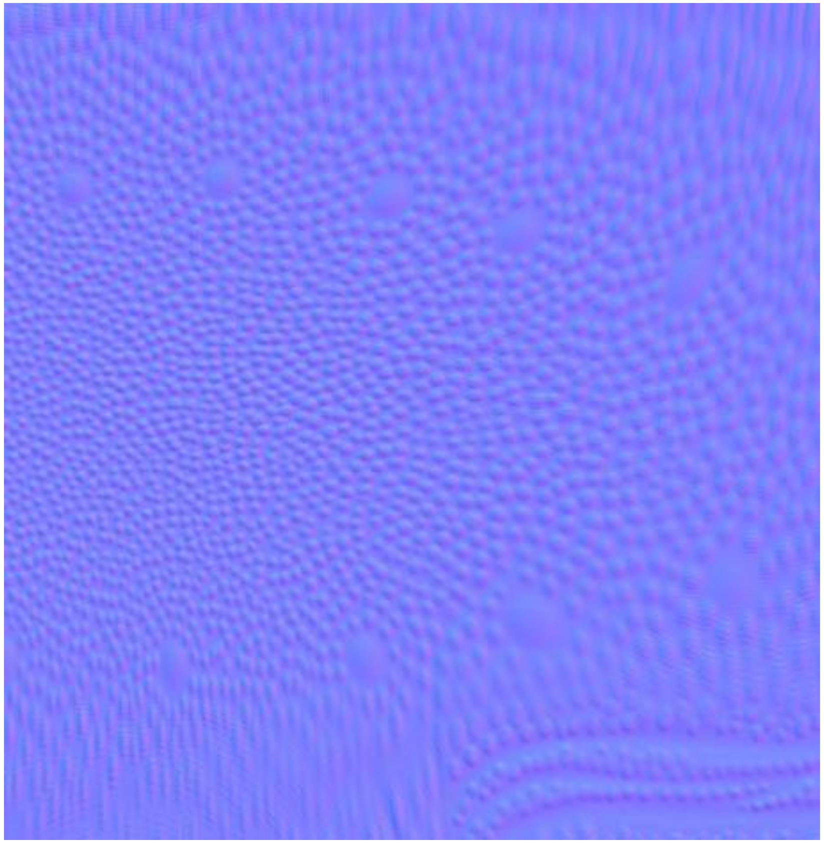
  - **Note:** N' is not normalized
- Bump map code similar to normal map code.
- Just compute, use **n'** instead of **n**

Bump mapping: Blinn's method

- **Option b (right):** Store values of u, v as a heightfield
  - Slope of consecutive columns determines how much **n** along u
  - Slope of consecutive rows determines how much **n** along v
- **Option c (Angel textbook):** Encode using differential equations

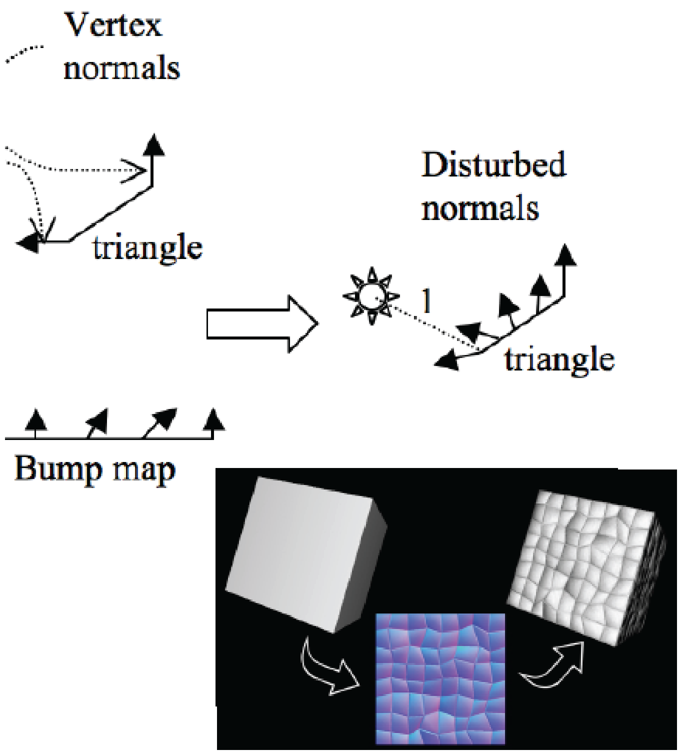


Bump mapping: examples



Bump Mapping Vs Normal Mapping

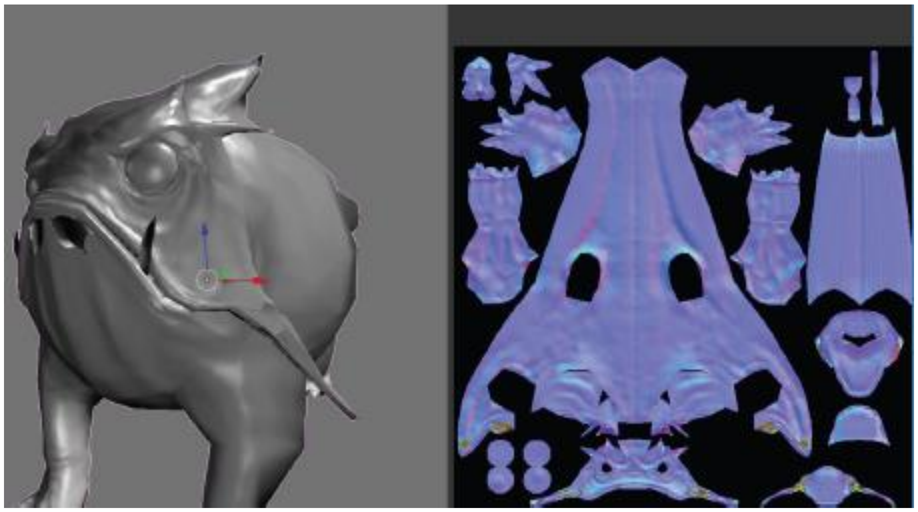
- **Bump mapping**
- (Normals $\mathbf{n}=(n_x, n_y, n_z)$ stored as *local distortion of face orientation*. Same bump map can be tiled/repeated and reused for many faces)



- **Normal mapping**
- Coordinates of normal (relative to tangent space) are encoded in color channels
- Normals stored combines

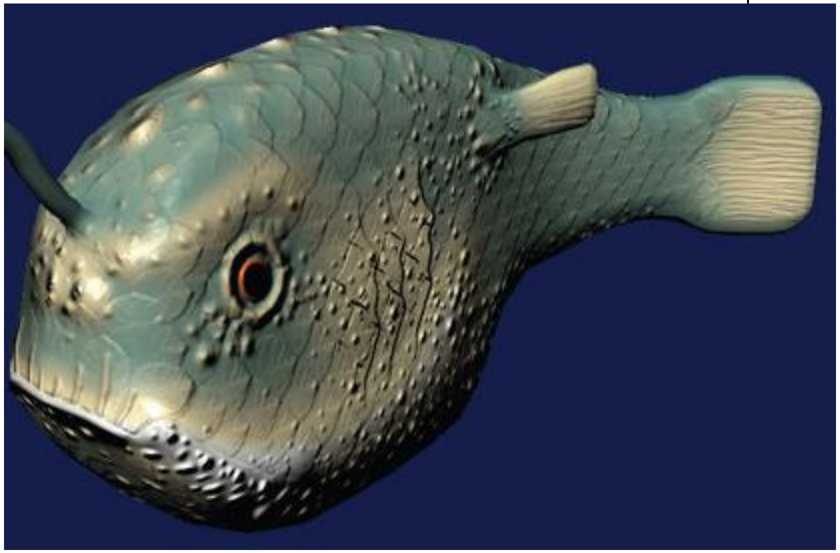
  face orientation + plus distortion.

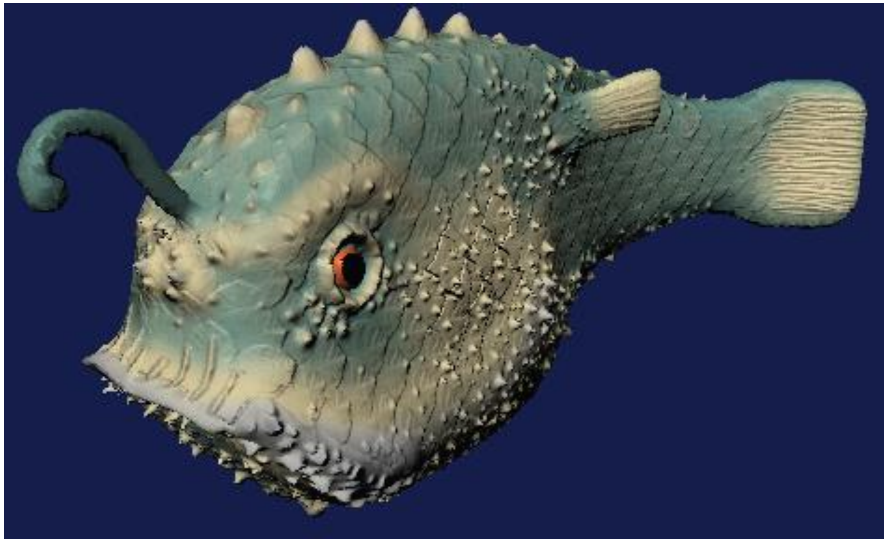
)



Displacement Mapping

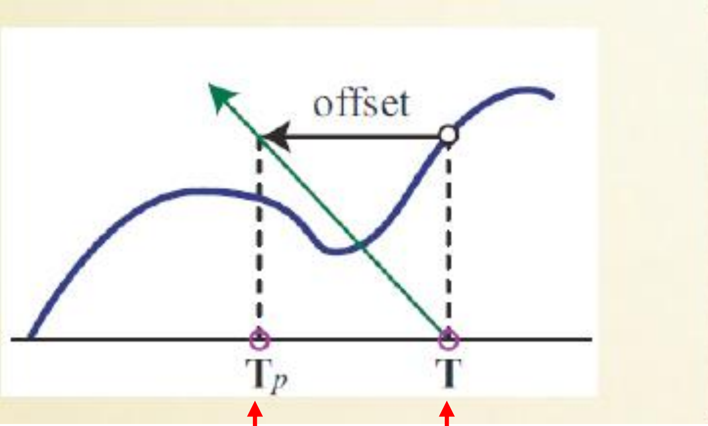
- Uses a map to displace the surface at each position
- Offsets the position per pixel or per vertex
  - Offsetting per vertex is easy in vertex shader
  - Offsetting per pixel is architecturally hard





Parallax Mapping

- Bump and normal maps increase surface detail, but do not simulate:
  - Parallax effects: Slanting of texture with view angle
  - Blockage of one part of surface by another part
- Parallax mapping
  - simulates parallax effects
  - Looks up a texture location offset depending on view angle
  - Different texture returned after offset

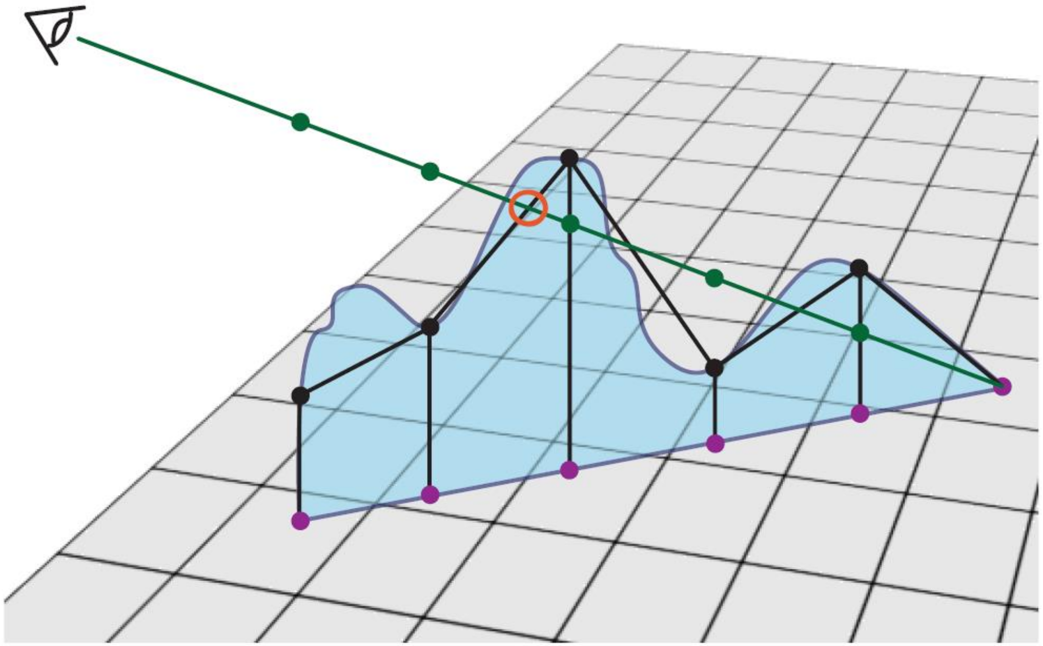


Parallax map
Looks up here

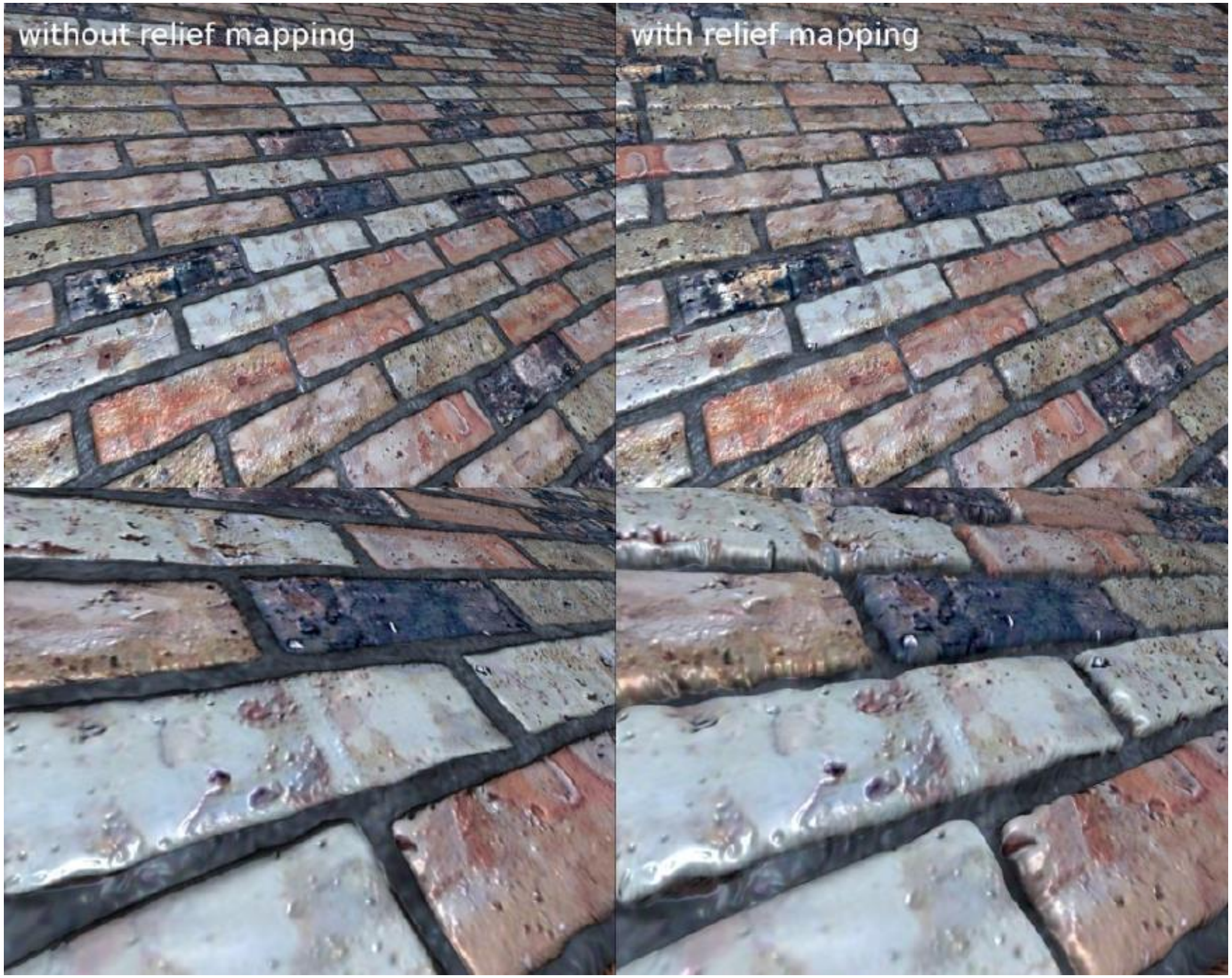
Normal map
Looks up here

Relief (or Parallax Occlusion) Mapping

- Parallax mapping approximates parallax
- Sometimes doesn't work well for occlusion effects
- Implement a heightfield raytracer in a shader, detect blockage
- Pretty expensive, but looks amazing



Relief Mapping Example

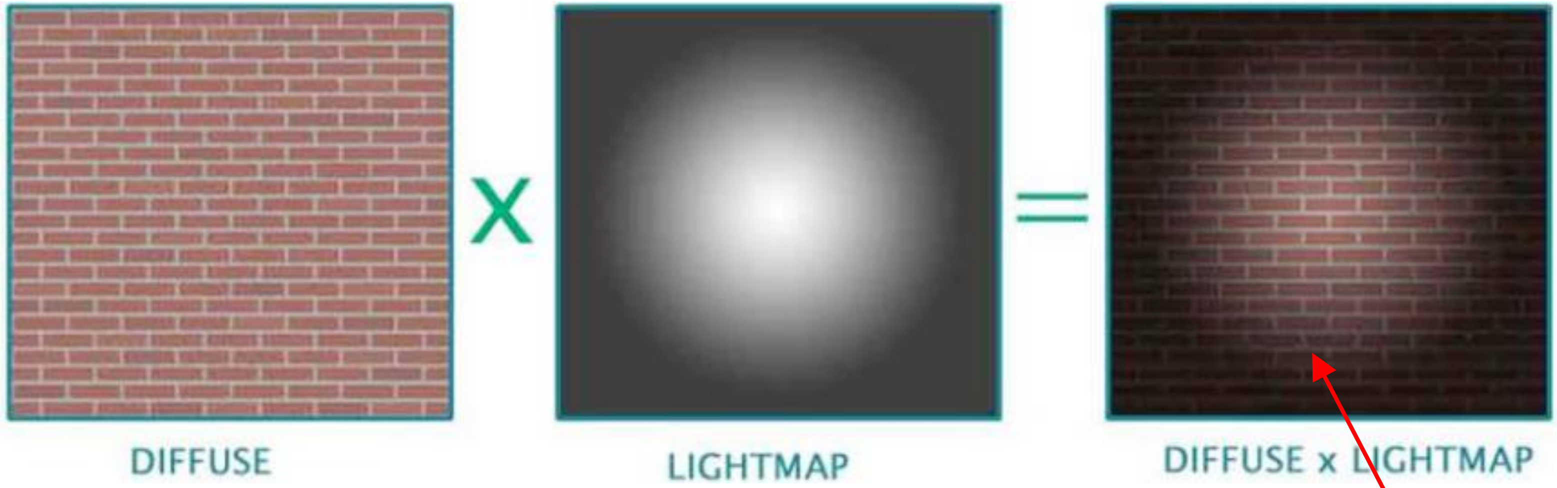


Cool YouTube Video: https://youtu.be/EkLKhsRzE-g

Light Mapping

Light Maps

- Good shadows are complicated and expensive
- If light and object positions do not change, shadows do not change
- Can “bake” the shadows into a texture map as a preprocess step
- During lighting, lightmap values are multiplied into resulting pixel



Apply this in fragment shader

Specular Mapping

- Store specular in a map
- Use greyscale texture as a multiplier for specular component

Alpha Mapping

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

Render Bush
on 1 polygon

Render Bush
on polygon rotated
90 degrees

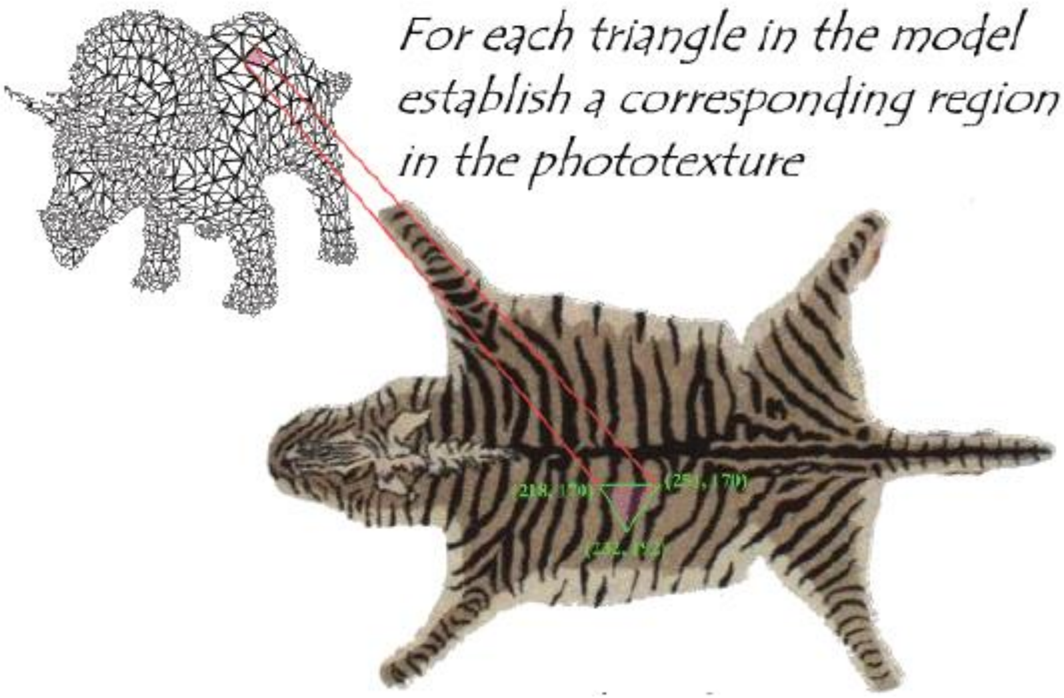
Alpha Mapping

- Rotation trick works at eye level (left image)
- Breaks down from above (right image)

Mesh Parametrization

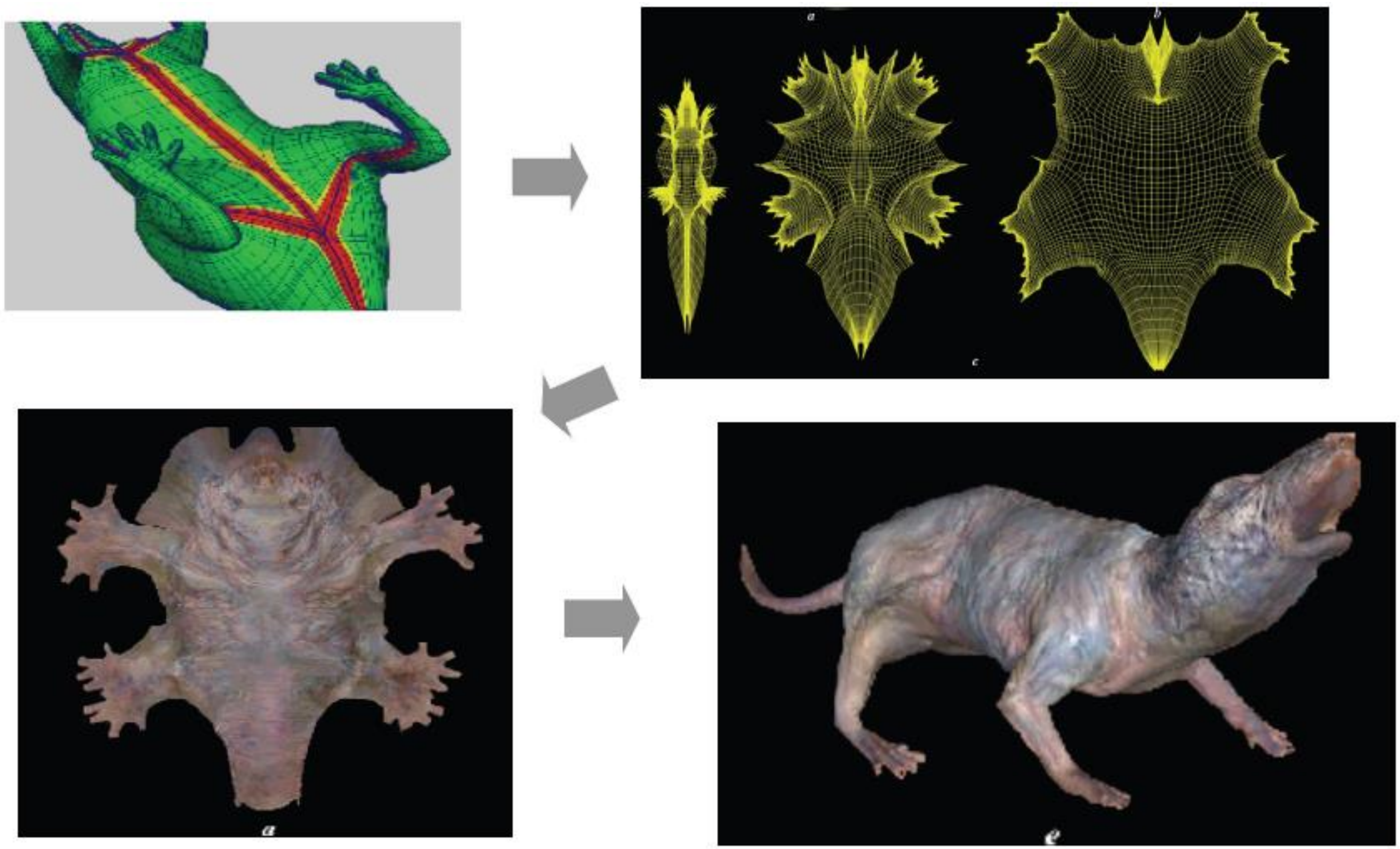
Mesh Parametrization

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



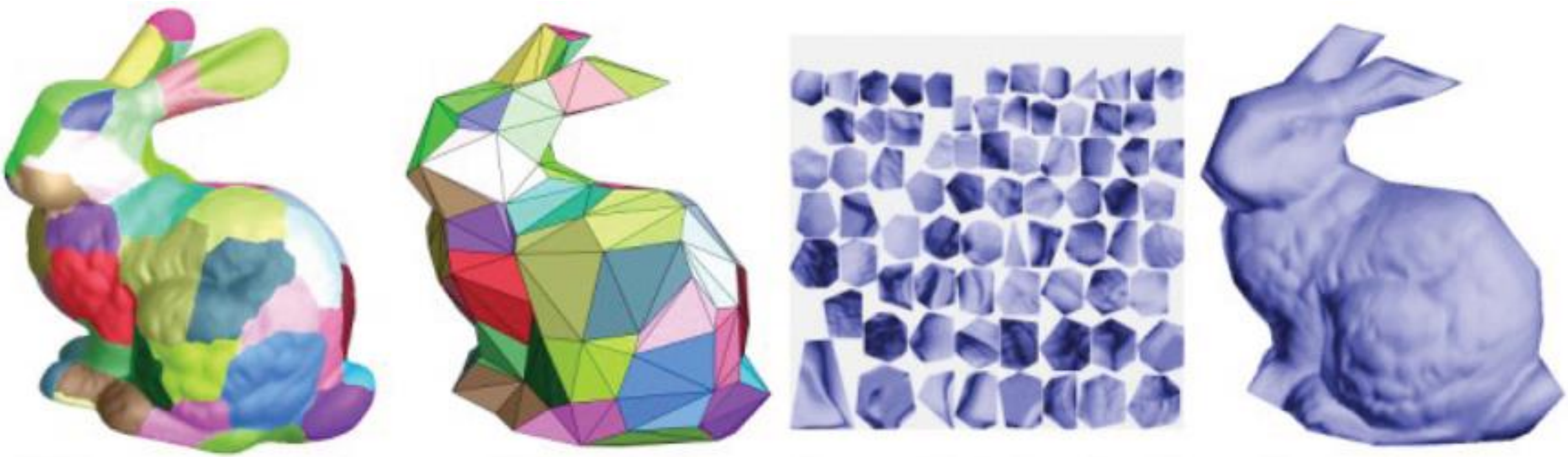
Parametrization in Practice

- 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

References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley
- Real Time Rendering by Akenine-Moller, Haines and Hoffman