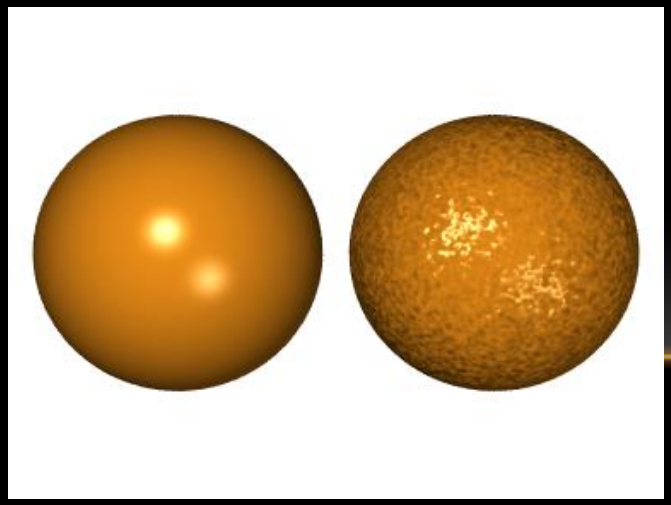
BUMP MAPPING

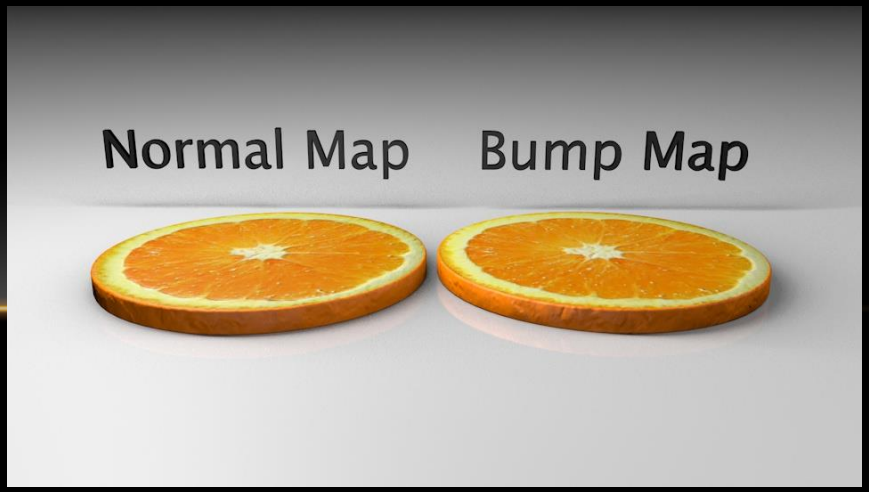
OUTLINE

- Bump Mapping
  - Procedural
  - Textural

MODELING AN ORANGE

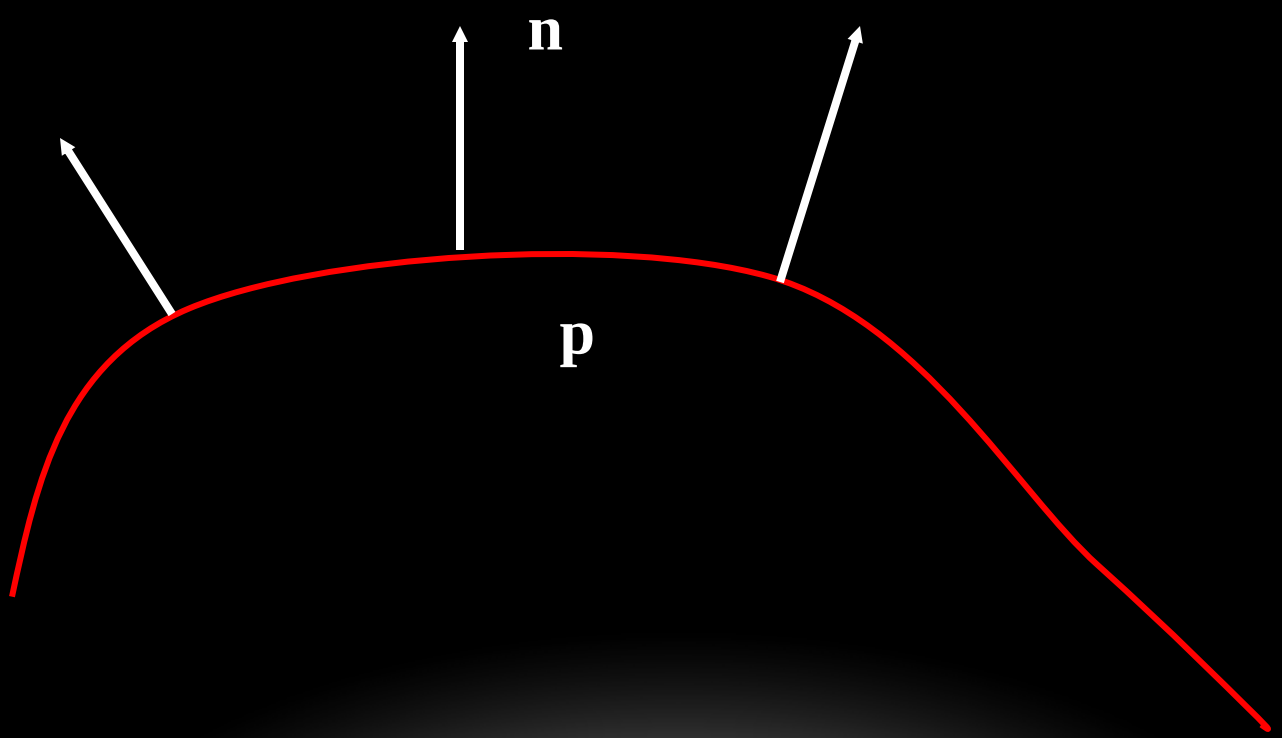
- Consider modeling an orange
- Texture map a photo of an orange onto a surface
  - Captures dimples
  - Will not be correct if we move viewer or light
  - We have shades of dimples rather than their correct orientation
- Ideally we need to perturb normal across surface of object and compute a new color at each interior point



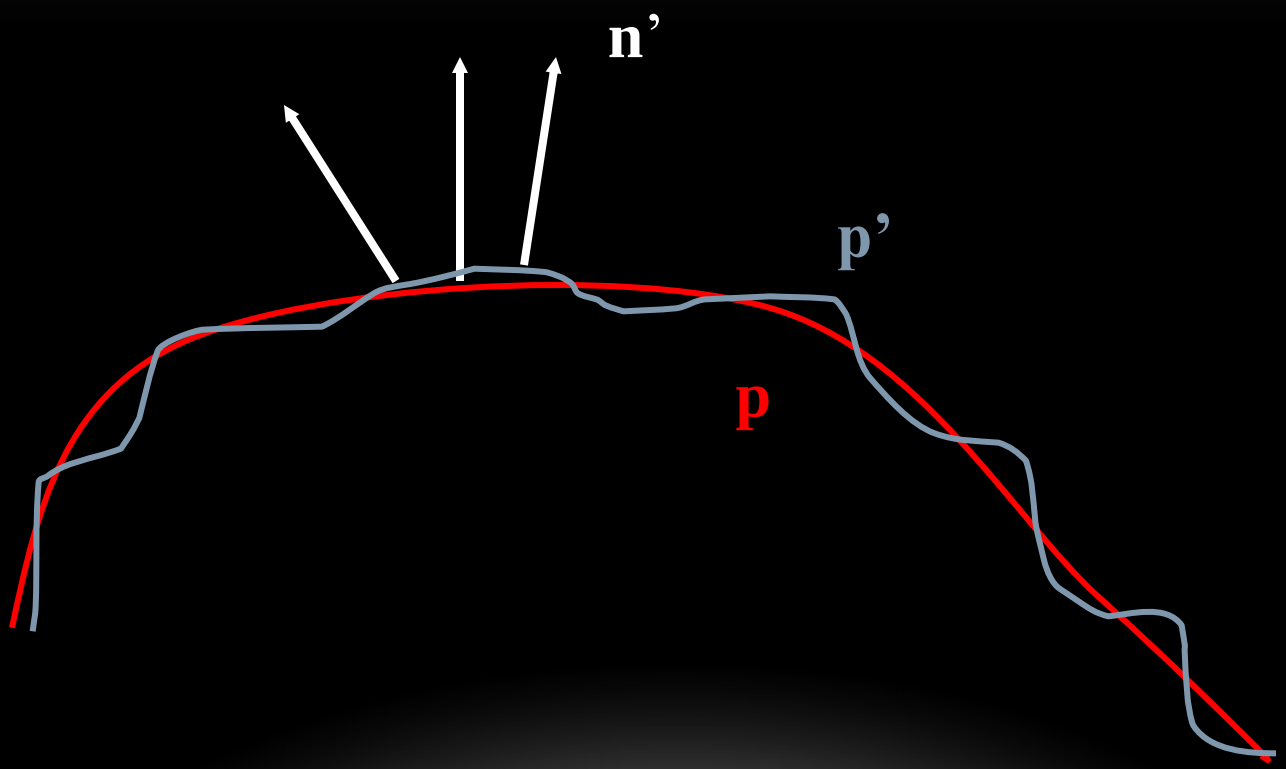


BUMP MAPPING (BLINN)

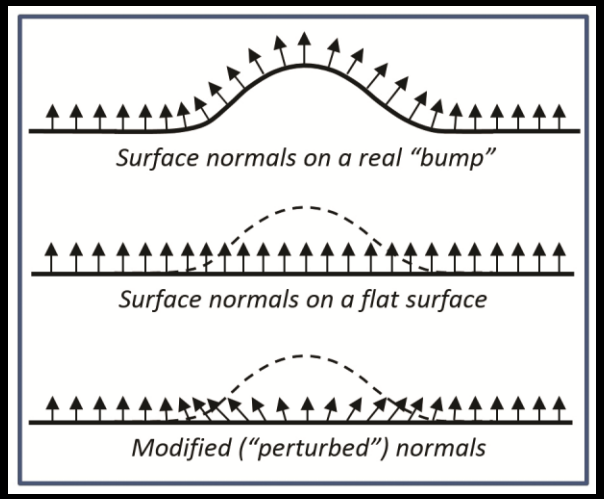
- Consider a smooth surface



ROUGHER VERSION

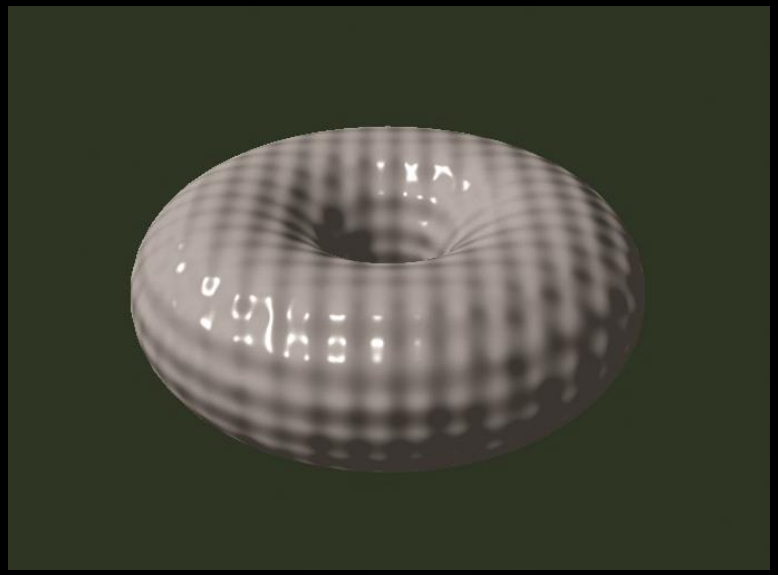


PROCEDURAL BUMP MAPPING



BUMP MAPPING EXAMPLE

- Perturbing the normals with a sine wave function for the torus gives us:



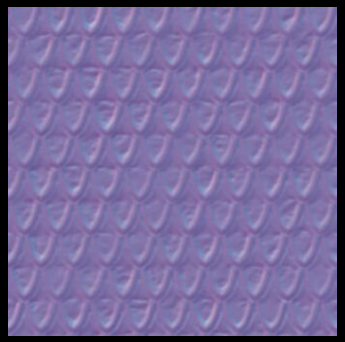
FRAGMENT SHADER CHANGES

```
void main(void)
{
    // normalize the light, normal, and view vectors:
    vec3 L = normalize(varyingLightDir);
    vec3 N = normalize(varyingNormal);
    vec3 V = normalize(-varyingVertPos);

    float a = 0.25;// controls depth of bumps
    float b = 100.0;// controls width of bumps
    float x = originalVertex.x;
    float y = originalVertex.y;
    float z = originalVertex.z;
    N.x = varyingNormal.x + a*sin(b*x);
    N.y = varyingNormal.y + a*sin(b*y);
    N.z = varyingNormal.z + a*sin(b*z);
    N = normalize(N);
…
```

```
…
    // compute light reflection vector, with respect N:
    vec3 R = normalize(reflect(-L, N));

    // get the angle between the light and surface normal:
    float cosTheta = dot(L,N);

    // angle between the view vector and reflected light:
    float cosPhi = dot(V,R);

    // compute ADS contributions (per pixel):
    fragColor = globalAmbient * material.ambient
                + light.ambient * material.ambient
                + light.diffuse * material.diffuse * max(cosTheta,0.0)
                + light.specular  * material.specular *
                pow(max(cosPhi,0.0), material.shininess);
}
```

NORMAL MAPPING

- Instead of using a procedural approach, use a look up table for normal perturbation
  - Can use a texture map for these purposes
  - Instead of storing r, g, b color values, store x, y, z values for normals
  - Instead of applying texels to color, we use them to modify normals



CONVERTING XYZ NORMAL TO RGB

- RGB values are positive, but XYZ normals can be positive or negative
  - If we restrict our XYZ values to between -1…+1 we can convert by:

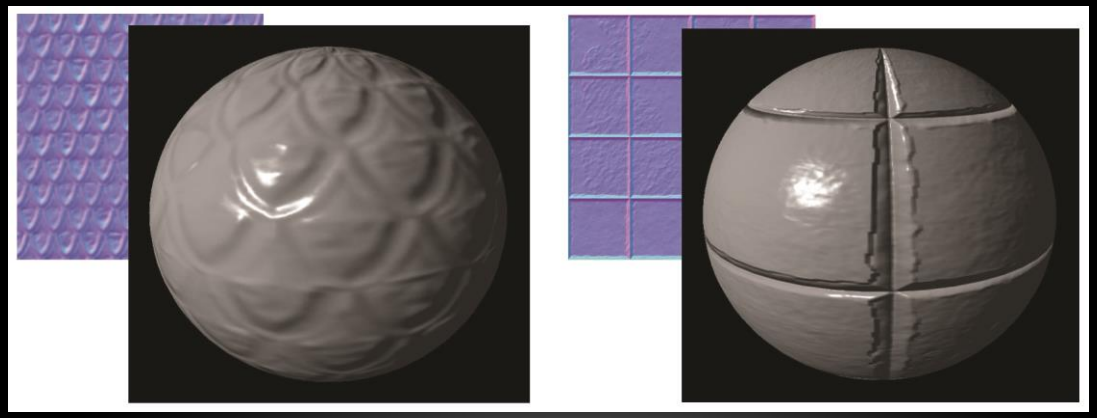
$$R = (N_X + 1)/2$$

$$G = (N_Y + 1)/2$$

$$B = (N_Z + 1)/2$$

STORING NORMALS

- In normal map, normals are represented with respect to an arbitrary x-y plane
  - Values stored in a texture map are their deviations from “vertical” with respect to this plane, z component set to 1
    - e.g. A perpendicular normal would be [0, 0, 1], so stored as [.5 .5 1]

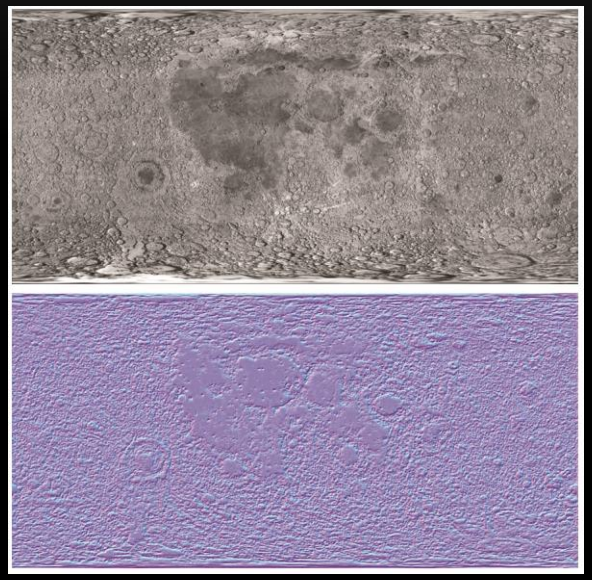


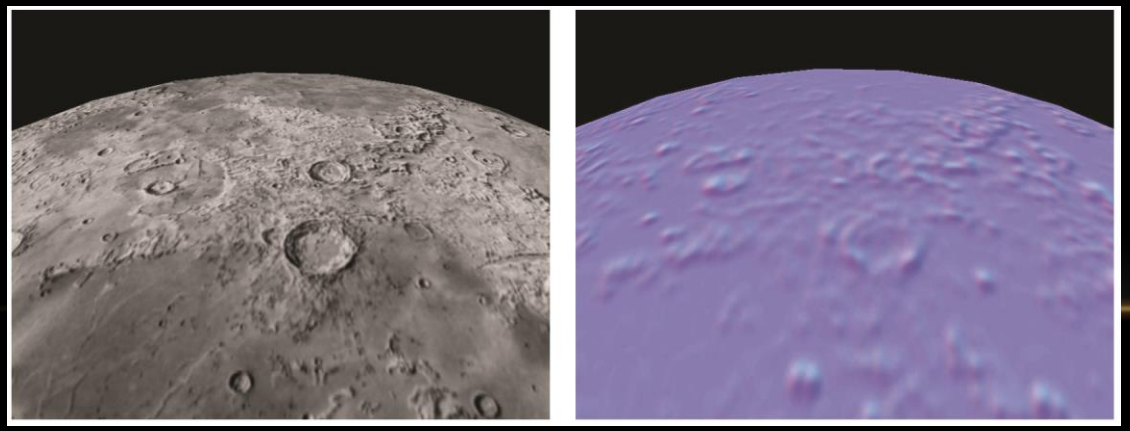
USING NORMAL MAP

- We need to build a transformation matrix from the arbitrary x-y plane to the camera space
  - At each object vertex, use a plane that is tangent to the object
  - Define two mutually perpendicular vectors in that plane, also perpendicular to the normal
    - Called the tangent and bitangent (or binormal)
  - Can (fairly easily) get the tangent, then the bitangent is just the cross product of the tangent and normal
  - If the tangent is not able to be analytically derived, can approximate by using vectors between vertices
- Tangents are sent to the shaders just like normals, etc.
- Use the TBN matrix to transform normals relative to the object
  - Need to convert from “rgb” values to xyz – multiply by 2 and subtract 1

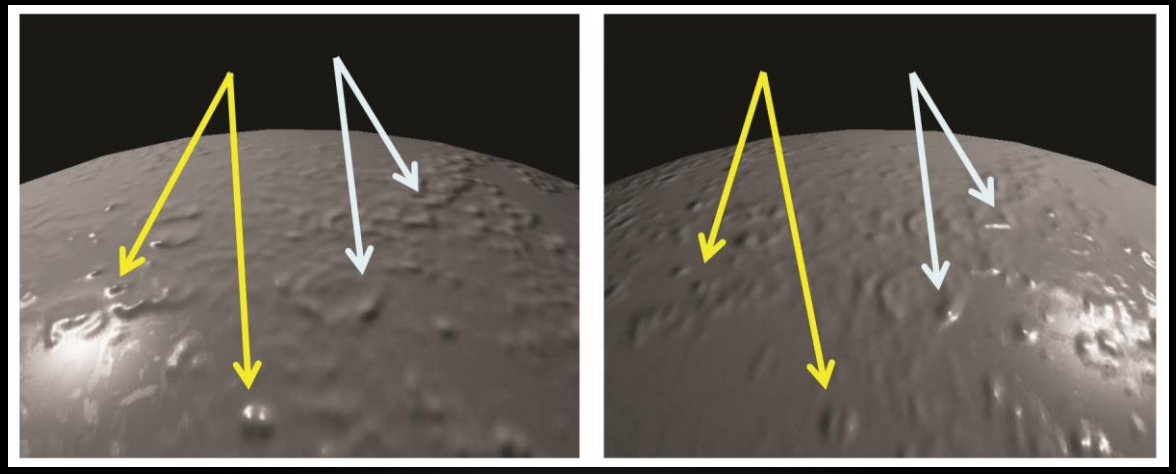
NORMAL MAP FROM A TEXTURE

- Tools exist to generate normals from texture maps
  - Use edge detection to infer peaks and valleys
- Both the texture and normal map can then be used to texture an object

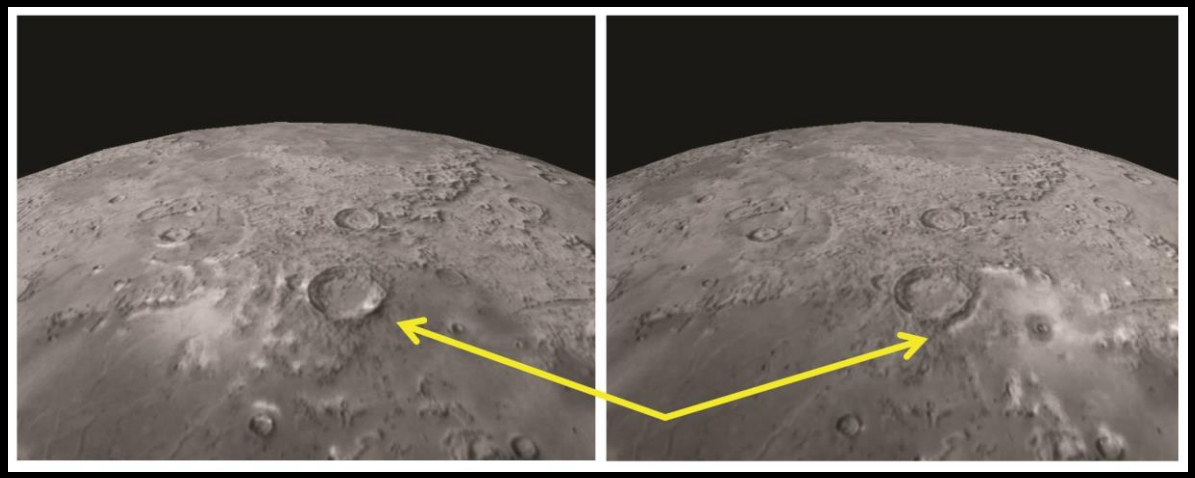




NORMAL MAPPING ALONE

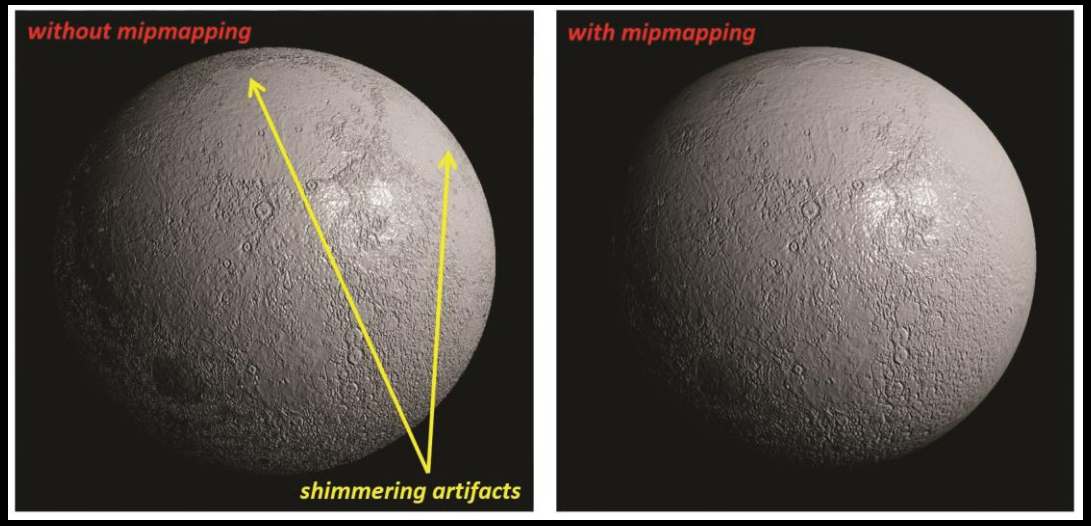


NORMAL MAPPING COMBINED WITH TEXTURE MAP

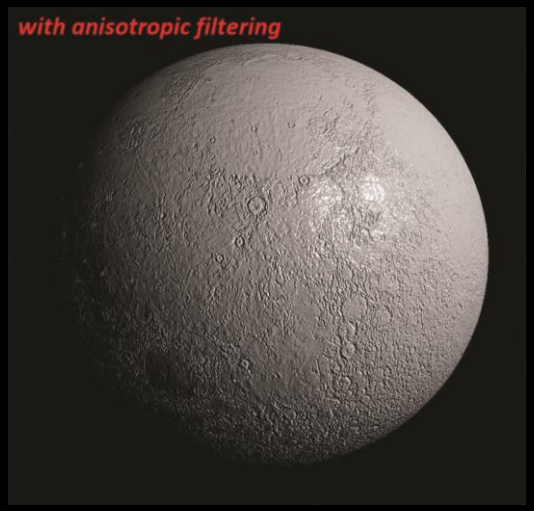


SAME PROBLEMS AS TEXTURING

- May need to use texturing fixes to avoid problems



CORRECTING ALIASING PROBLEMS



SUMMARY

- Bump Mapping
  - Procedural
  - Textural