Computer Graphics

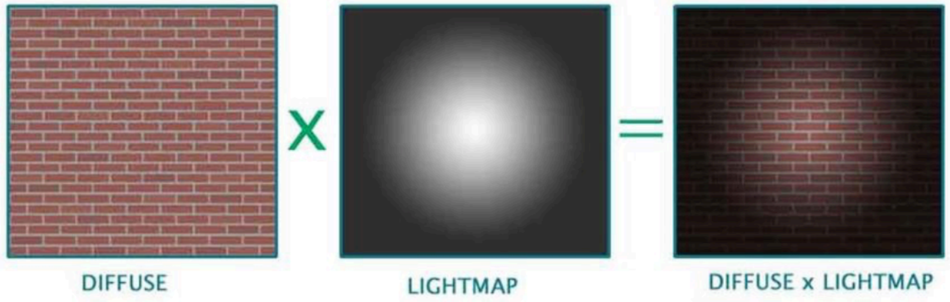
Texture Mapping Part 2

Mark Pauly

Geometric Computing Laboratory

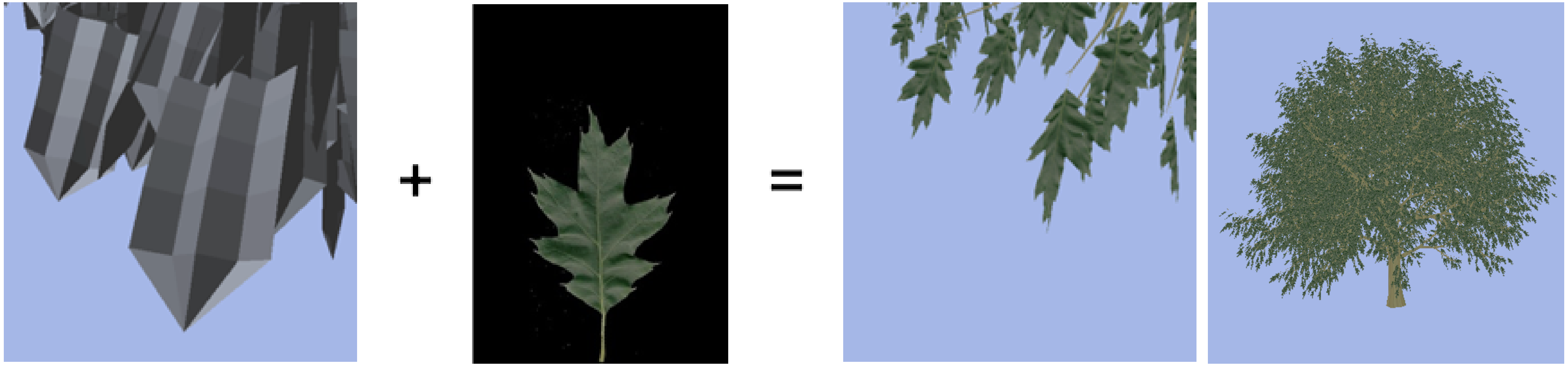
Special Texture Maps

Light Maps



Alpha Maps

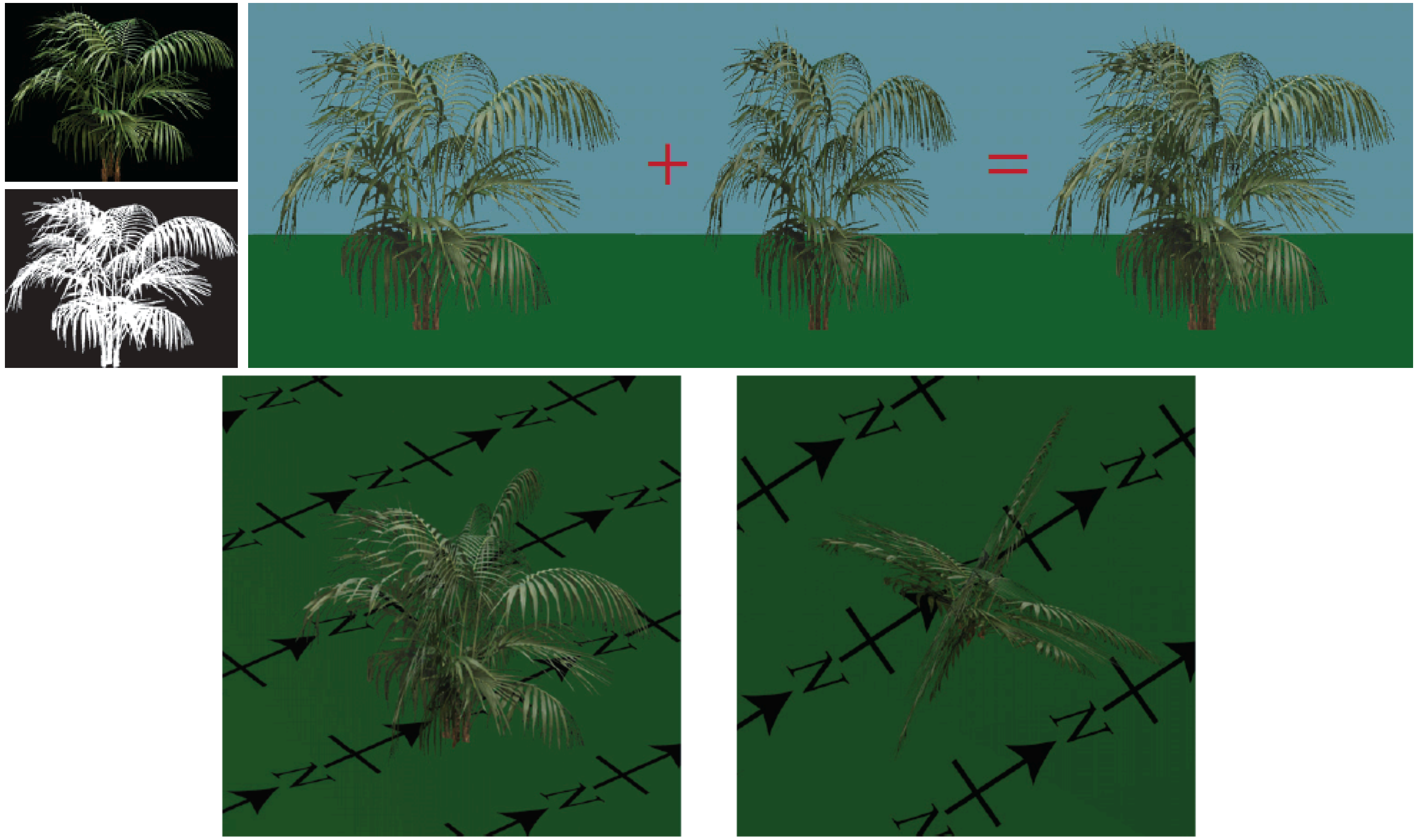
- Discard transparent texture pixels (alpha=0) in fragment shader.
- Frequently used for real-time rendering of plants.



Images courtesy of Oliver Deussen

Alpha Maps

- Discard transparent texture pixels (alpha=0) in fragment shader.
- Frequently used for real-time rendering of plants.



Images from Akenine-Möller et al., Real-Time Rendering

Demo: Alpha Maps

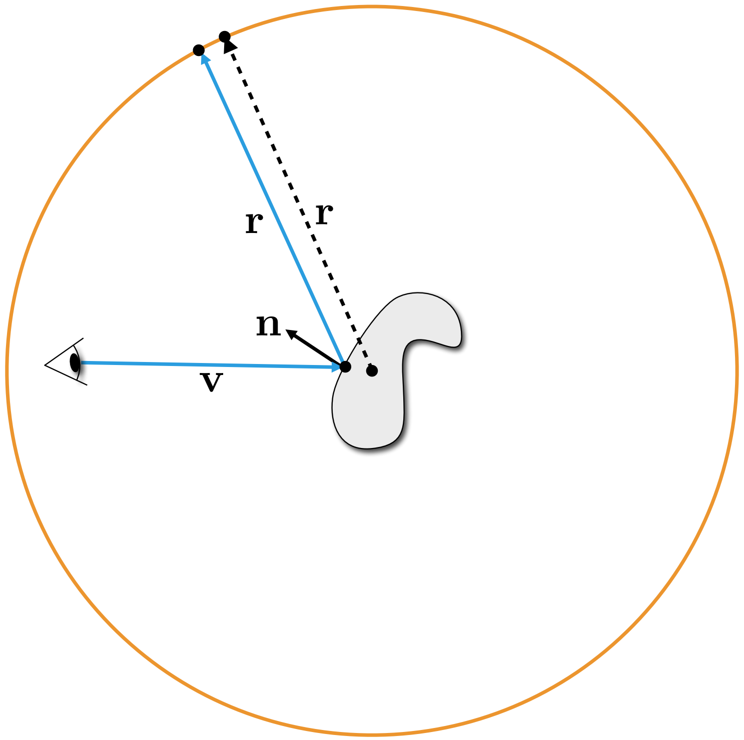
Press `Shift–Enter` to compile shaders

Environment Maps

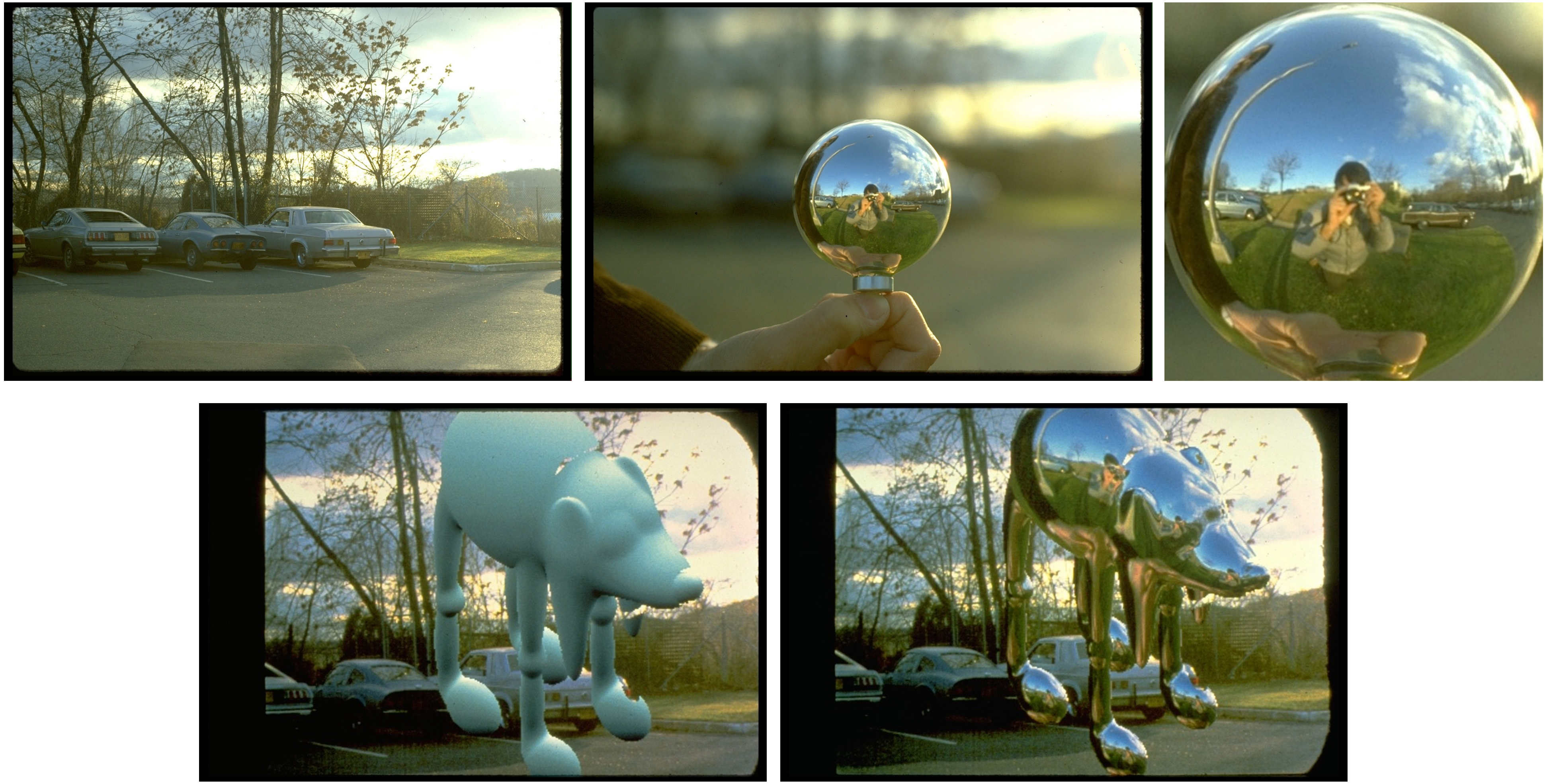
- Approximate reflections of environment at surface.
- Environment is assumed to be far away from object.

Spherical Environment Maps

- Reflect viewing direction to access texture
- Environment is assumed to be far away from object.
- Neglect intersection point, only use reflected direction (*dashed ray in image*)
- Texture access depends on view direction and normal only.
- Sometimes, view direction is assumed to be $(0, 0, -1)$.



Spherical Environment Maps



Images from Gene Miller

Spherical Environment Maps

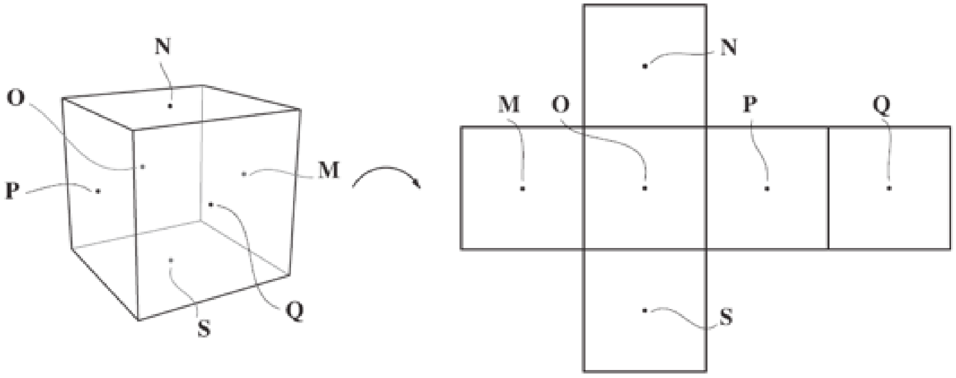
Environment Map

Cat with environment map

Cube Environment Maps

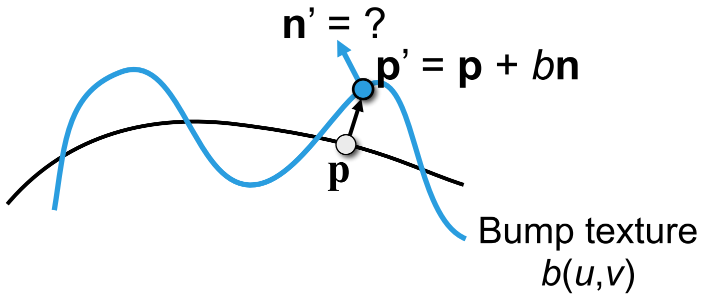
👍 Cube maps are a better representation, since they have less distortion.

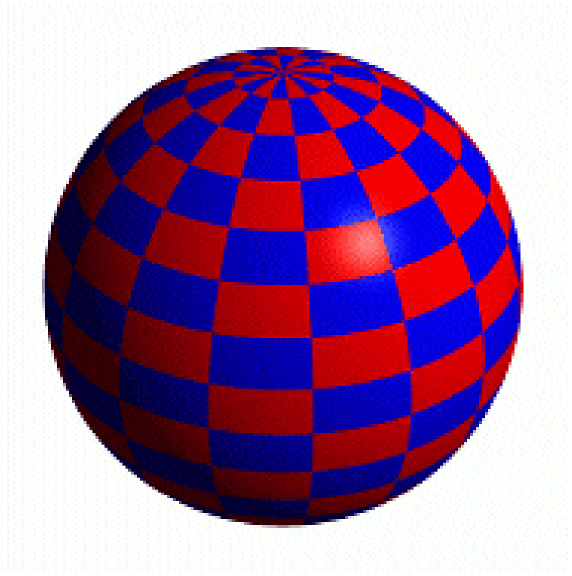
👎 However, they have to store six images.

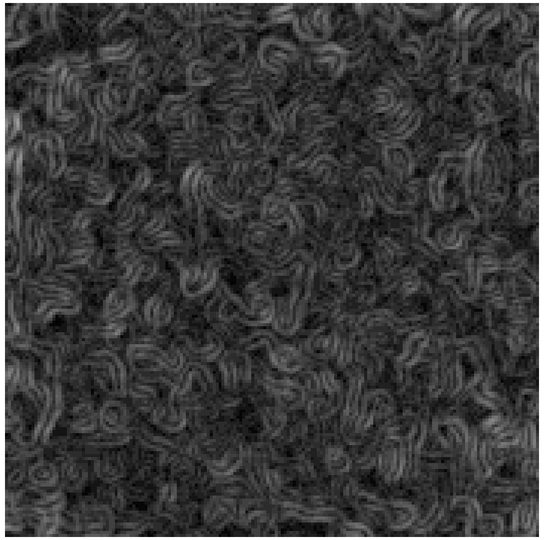


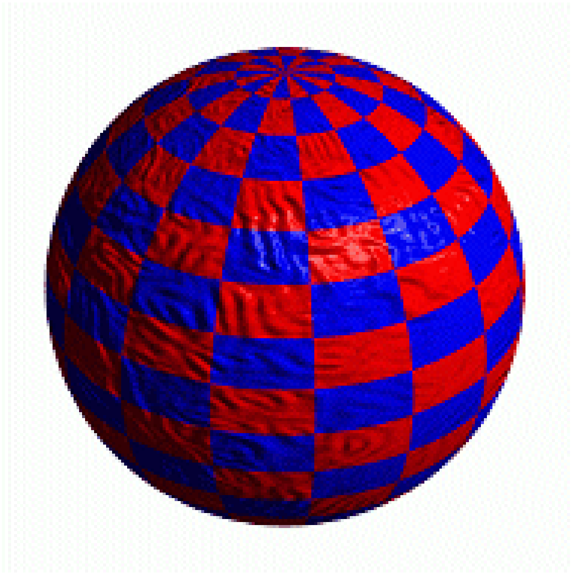
Bump Maps

- Add surface detail without increasing geometric complexity
- Assume bumps are small compared to geometry
- Perturb surface normal before lighting computations
- Bump pattern is taken from a texture







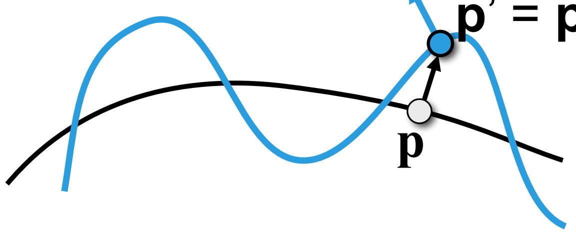


standard rendering | *bump map* | *bump-mapped result*

How to compute perturbed normal?

$$\mathbf{p}'(u,v) = \mathbf{p}(u,v) + b(u,v)\mathbf{n}(u,v)$$

$$\mathbf{n}(u,v) = \underbrace{\frac{\partial \mathbf{p}(u,v)}{\partial u}}_{\text{Tangent } \mathbf{t}} \times \underbrace{\frac{\partial \mathbf{p}(u,v)}{\partial v}}_{\text{Bitangent } \mathbf{b}}$$



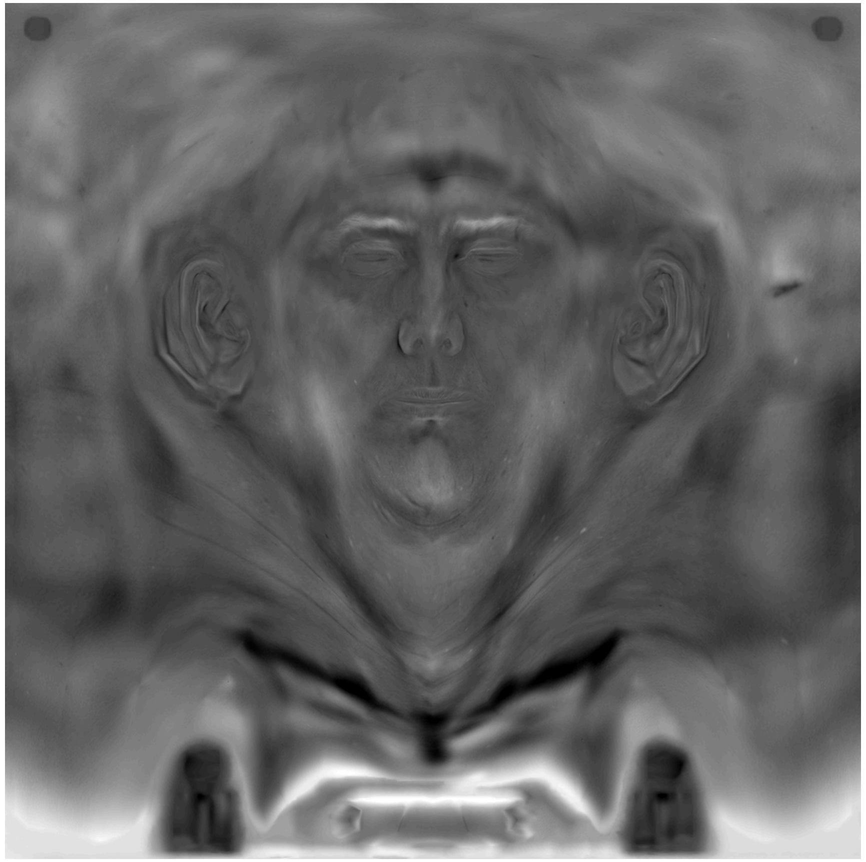
$$\frac{\partial \mathbf{p}'(u,v)}{\partial u} = \frac{\partial \mathbf{p}(u,v)}{\partial u} + \frac{\partial b(u,v)}{\partial u}\mathbf{n}(u,v) + b(u,v)\frac{\partial \mathbf{n}(u,v)}{\partial u} \approx \frac{\partial \mathbf{p}(u,v)}{\partial u} + \frac{\partial b(u,v)}{\partial u}\mathbf{n}(u,v)$$

$$\frac{\partial \mathbf{p}'(u,v)}{\partial v} \approx \frac{\partial \mathbf{p}(u,v)}{\partial v} + \frac{\partial b(u,v)}{\partial v}\mathbf{n}(u,v)$$

$$\mathbf{n}'(u,v) = \frac{\partial \mathbf{p}'(u,v)}{\partial u} \times \frac{\partial \mathbf{p}'(u,v)}{\partial v} \approx \mathbf{n} + \frac{\partial b}{\partial u}(\mathbf{n} \times \mathbf{b}) + \frac{\partial b}{\partial v}(\mathbf{t} \times \mathbf{n})$$

Can be computed from discrete height field using finite differences.

Bump Maps



Bump map for head scan

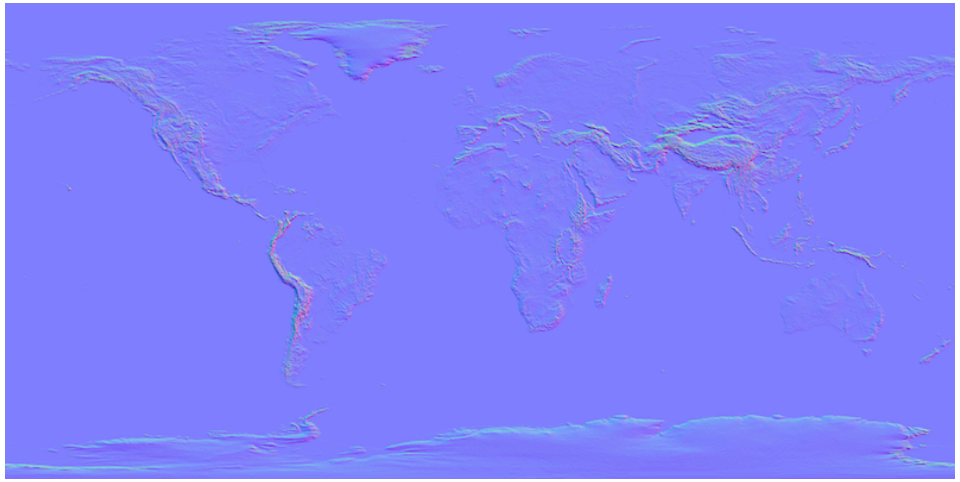
Bump-mapped mesh

3D Head Scan by Lee Perry-Smith, based on work at TripleGangers

Bump Maps vs Normal Maps

- **Bump mapping**
  - Perturb normal based on height field texture
  - Tangents and bitangents can be approximated in pixel shader (see here)
  - Have to store one channel only
- **Normal mapping**
  - Store normal perturbation in RGB texture (need three channels)
  - Tangents and bitangents can be approximated in pixel shader (see here)
  - Accurate reproduction of normals
  - Generally preferred, unless memory is tight

Normal Maps

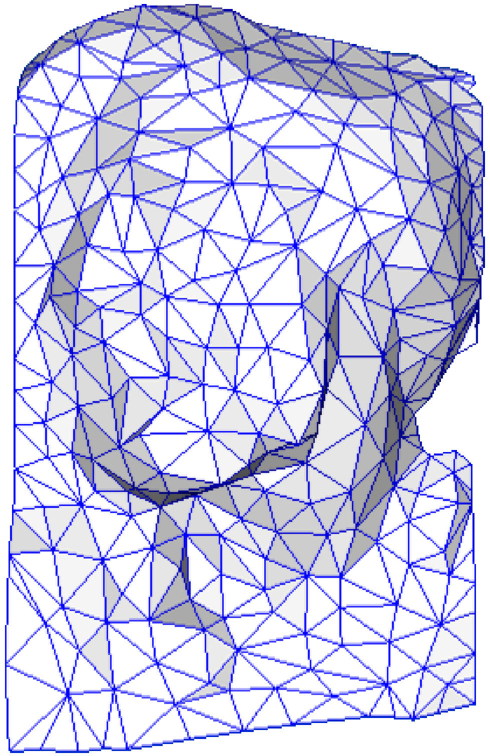


Normal map

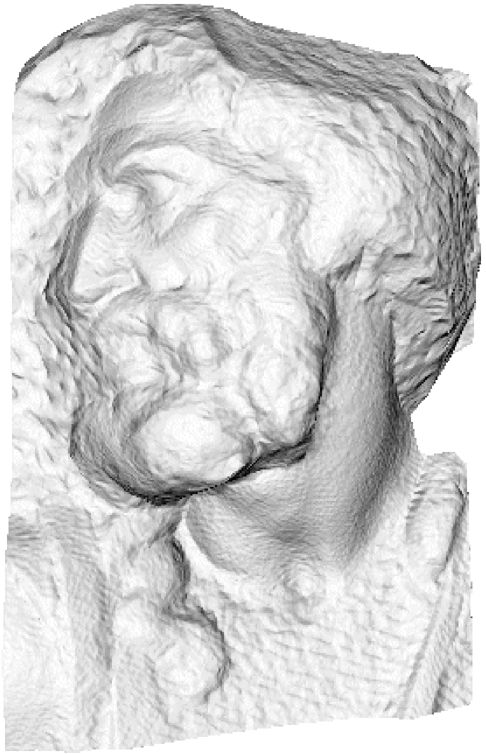
```
// normal, tangent, bitangent
vec3 N=..., T=..., B=...;
// access normal map
vec3 dn = texture(normal_map, texcoord.st).xyz;
// transform from [0,1] to [-1,1]
dn = 2.0 * dn - vec3(1,1,1);
// perturb normal in TBN coordinates
N = normalize(dn.x*T + dn.y*B + dn.z*N);
```

Normal Maps

original mesh
4M triangles



simplified mesh
500 triangles



simplified mesh
and normal mapping
500 triangles

Image courtesy Paolo Cignioni

Normal Maps

Demo by Timo Menzel

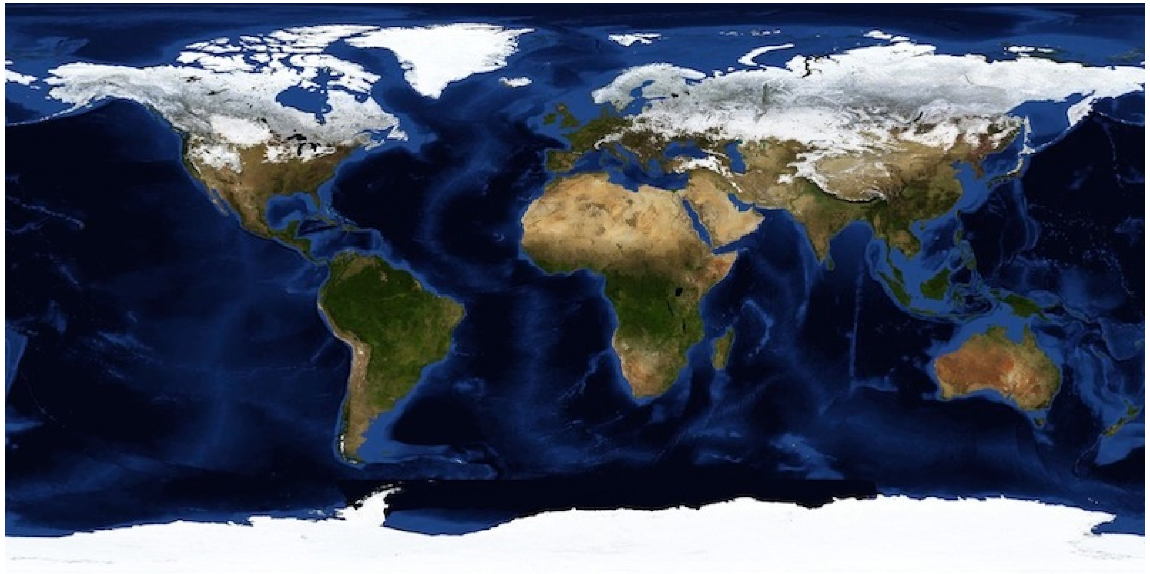
Normal Maps vs. Displacement Maps

- **Normal mapping**
  - Don't change geometry, only change normals based on texture
  - Can be performed in pixel shader
  - Silhouette still looks wrong
- **Displacement mapping**
  - Displace vertices based on offset stored in texture
  - Compute normal vectors of displaced surface
  - Performed in geometry shader or tesselation shader - not supported by WebGL!
  - Silhouette looks ok, but much more expensive to compute

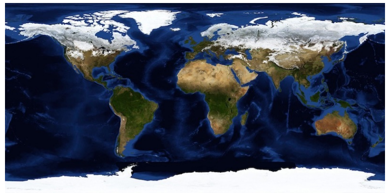
Multi-Texturing

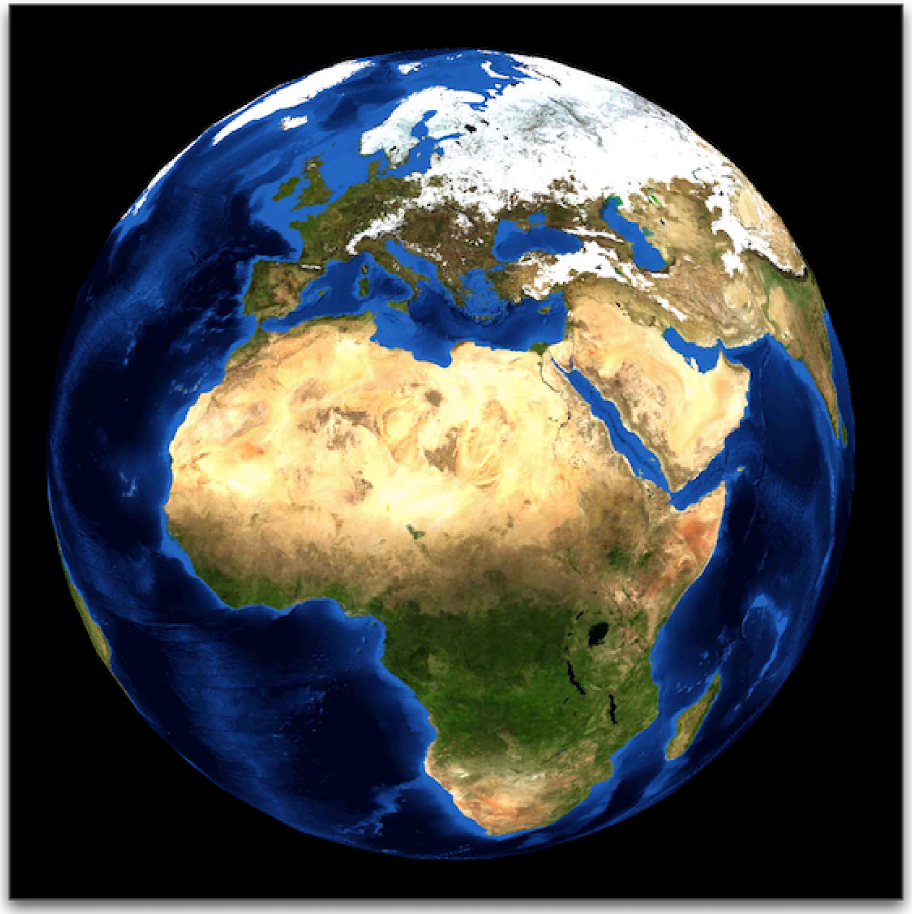
Nice Earth Textures

- NASA Blue Marble: Next Generation
  - Merged from many input images (from 2004)
  - Monthly day textures, night texture, clouds, ...

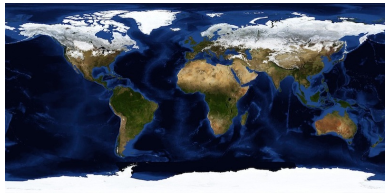


2D Texture

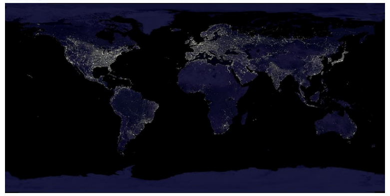




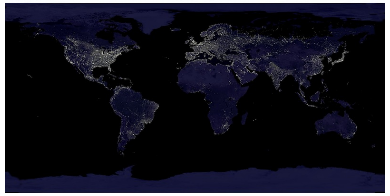
+ Diffuse and Specular Lighting

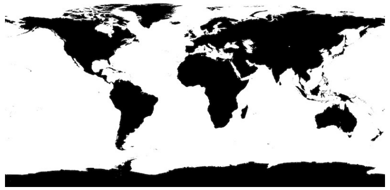


+ Night Texture

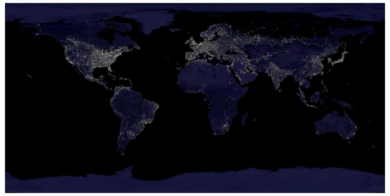
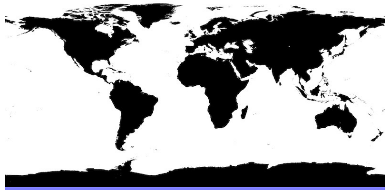
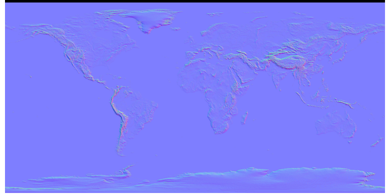


+ Specularity Map

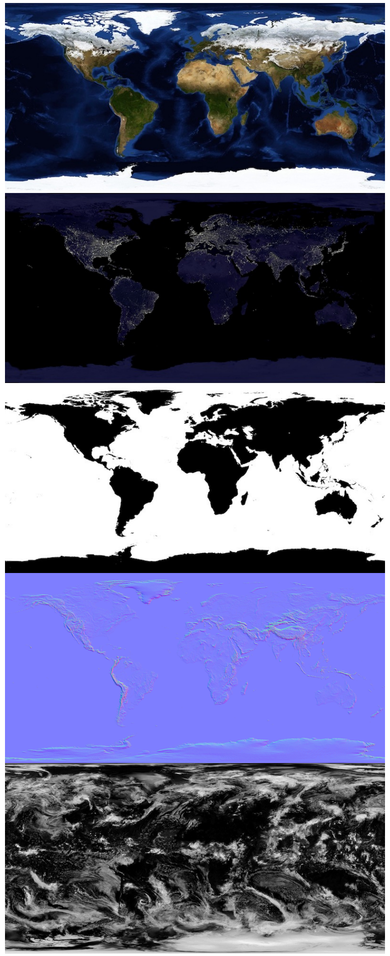




+ Normal Map


+ Clouds



Try it yourself!

Acknowledgments

Many thanks to Hartmut Schirmacher for providing aligned textures and initial WebGL code!

Prof. Hartmut Schirmacher,
Beuth Hochschule für Technik Berlin

Matching Quiz

Which antialising option or filter option solves which problem?

Bilinear filtering	Mipmapping	Anisotropic filtering	Fullscreen Antialiasing

...and drop them here into the correct category

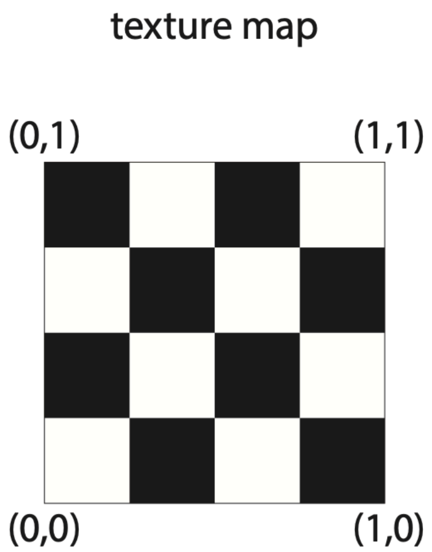
User zooms in	User zooms out	Triangles are seen from shallow angle	Small/thin geometric objects

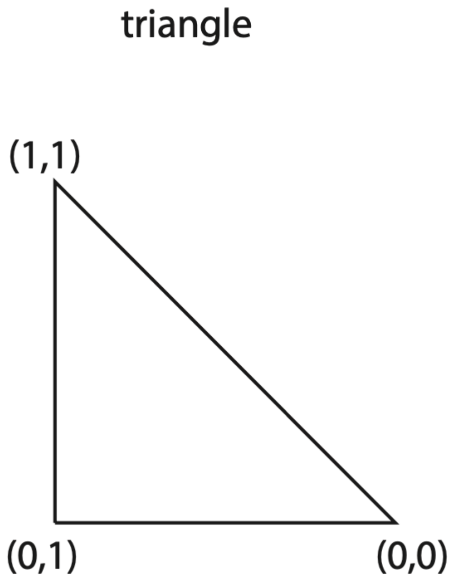
Summary

- **Texturing a triangle mesh**
  - Requires a parameterization or UV layout
  - Per-triangle barycentric interpolation of tex coords
- **Map per-pixel information onto surface meshes**
  - Material colors, opacity values
  - Reflection mapping, environment mapping
  - Normal vectors, vertex displacements
  - Combine effects with multi-texturing
- **Texture filtering is important**
  - Magnification: bilinear interpolation
  - Minification: mip-mapping, anisotropic filtering

Exam Question from 2024

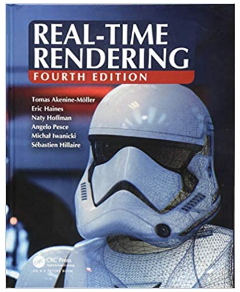
- **(4 points)** Given a texture map with a checkerboard pattern as illustrated below. We are rendering the triangle shown on the right using this texture with nearest neighbor texture sampling and no anti-aliasing. The corresponding texture coordinates are indicated at the vertices. Assume we are rendering the triangle under some arbitrary projective transformation. Let k be the percentage of rendered pixels of the triangle that are black. What range can k assume? Explain your answer.





Literature

- Akenine-Möller et al.: Real-Time Rendering, Taylor & Francis, 2021.
  - Chapter 6



- Marschner & Shirley: *Fundamentals of Computer Graphics*, 5th Edition, AK Peters, 2021.
  - Chapter 11