Approaching Photorealism in OpenGL

Lighting

Nathan Cournia and Andrew Van Pernis

{acnatha, arakel}@vr.clemson.edu.

Clemson University

Outline

Outline

- Motivation
- Basic OpenGL Lighting
- Advanced Lighting Techniques

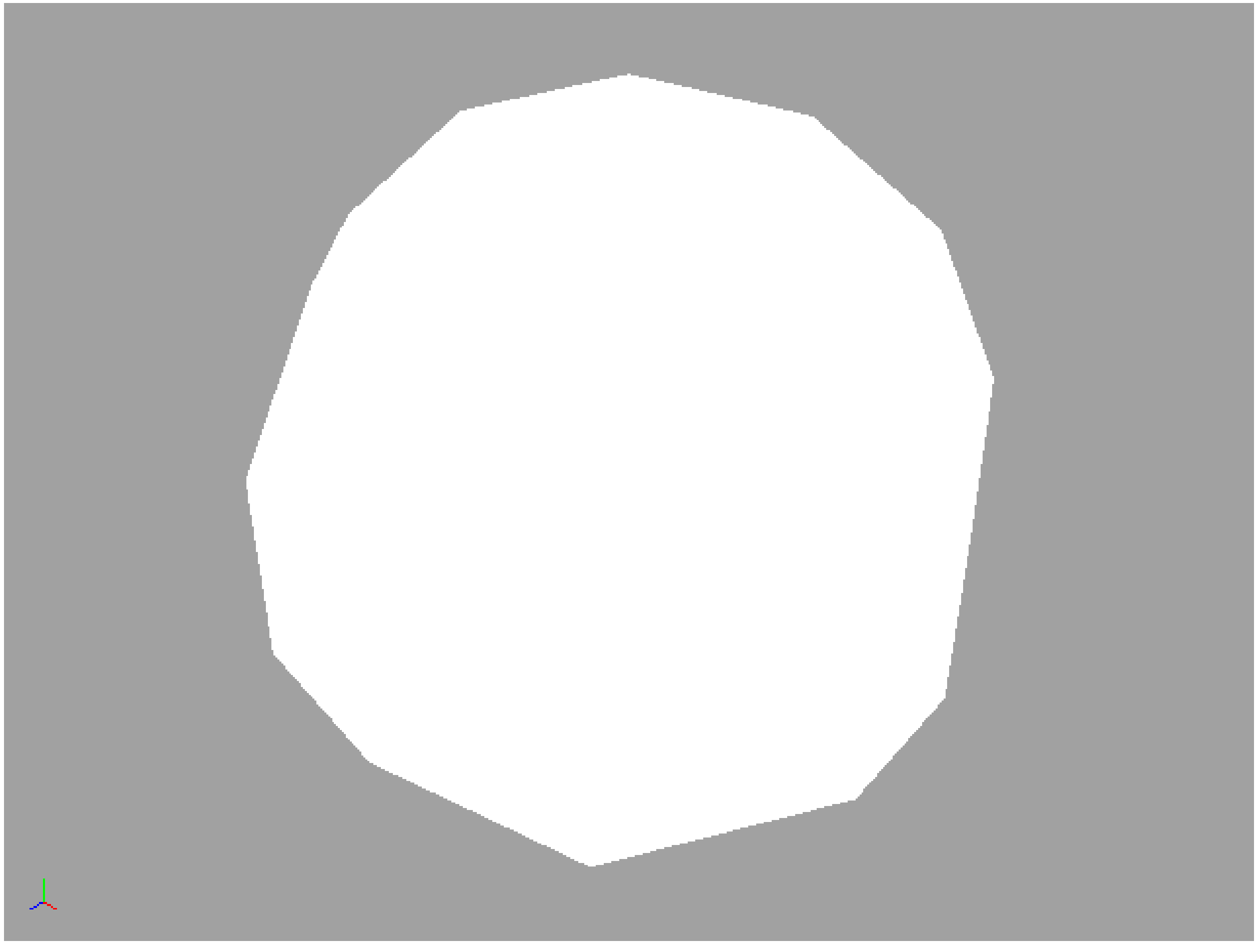
Motivation

Motivation

- Lighting provides visual cues for the user
  - Shape
  - Size
  - Texture
- Lighting is key for realism

Basic OpenGL Lighting

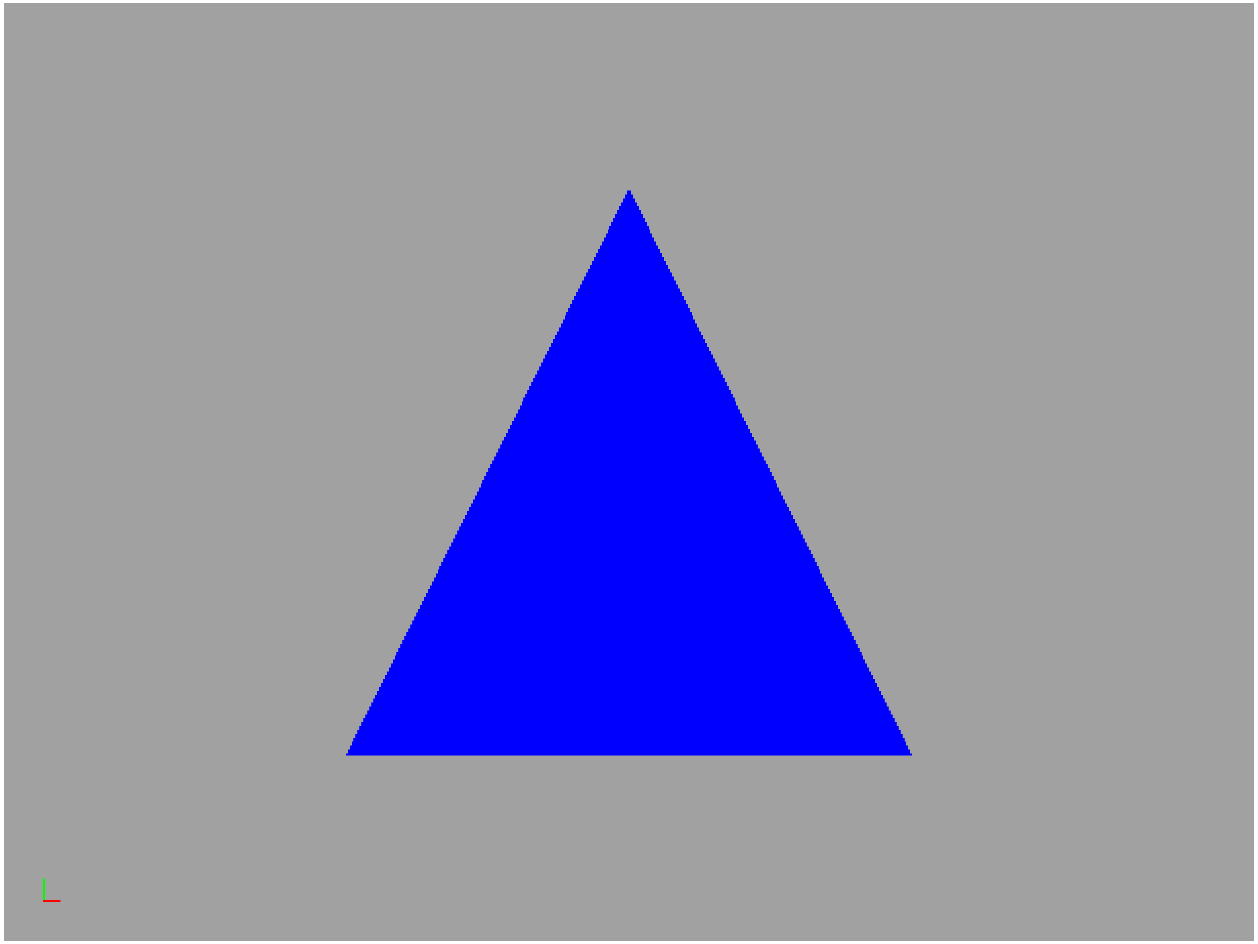
Without Lighting or Color



So Ugly

- Not practical
- Geometry data is not enough
- Needs color
- Needs lighting

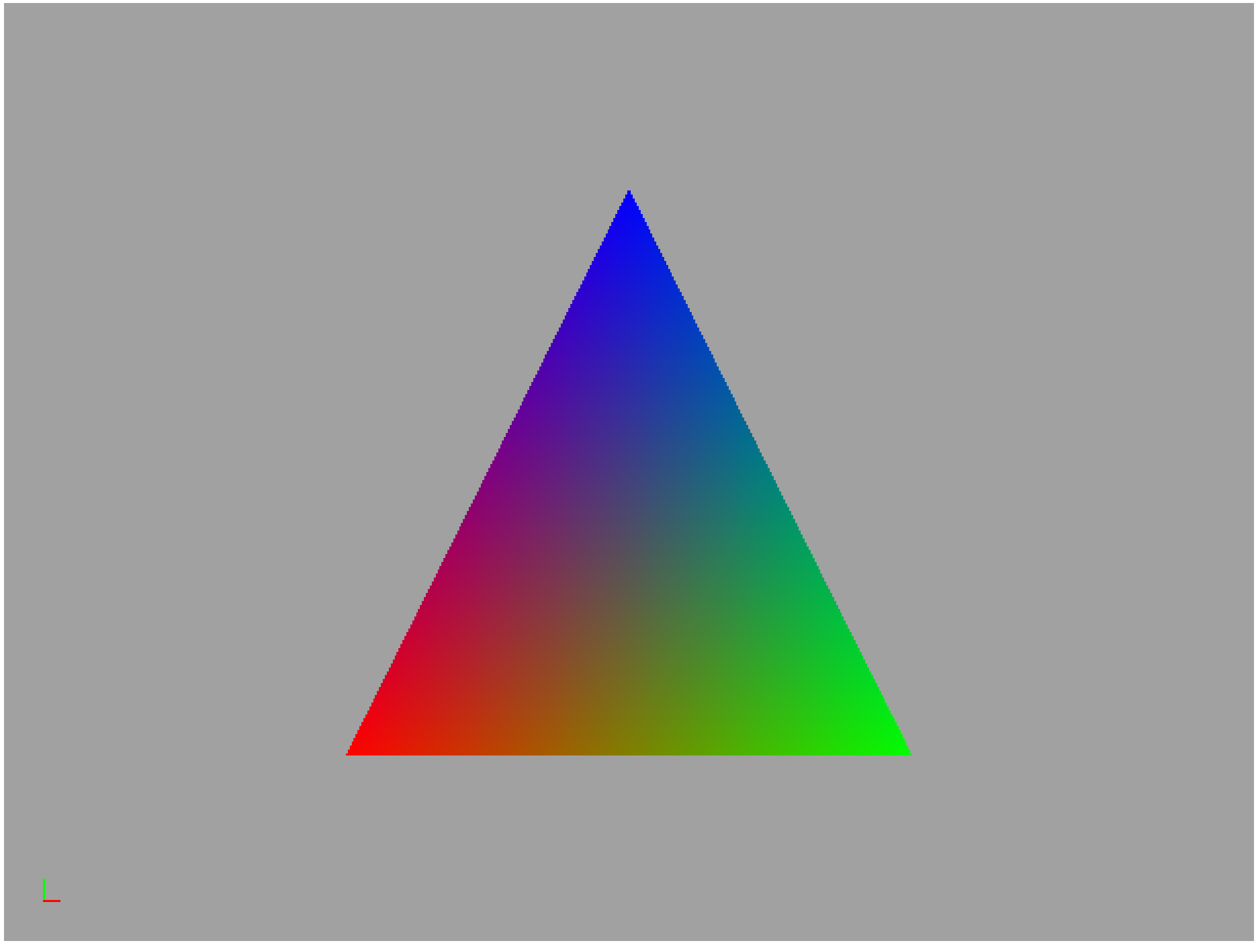
Flat Shading



Flat Shading (cont.)

- Each polygon is given a color
- Different colors can be assigned to different polygons
- No sense of depth

Smooth Shading



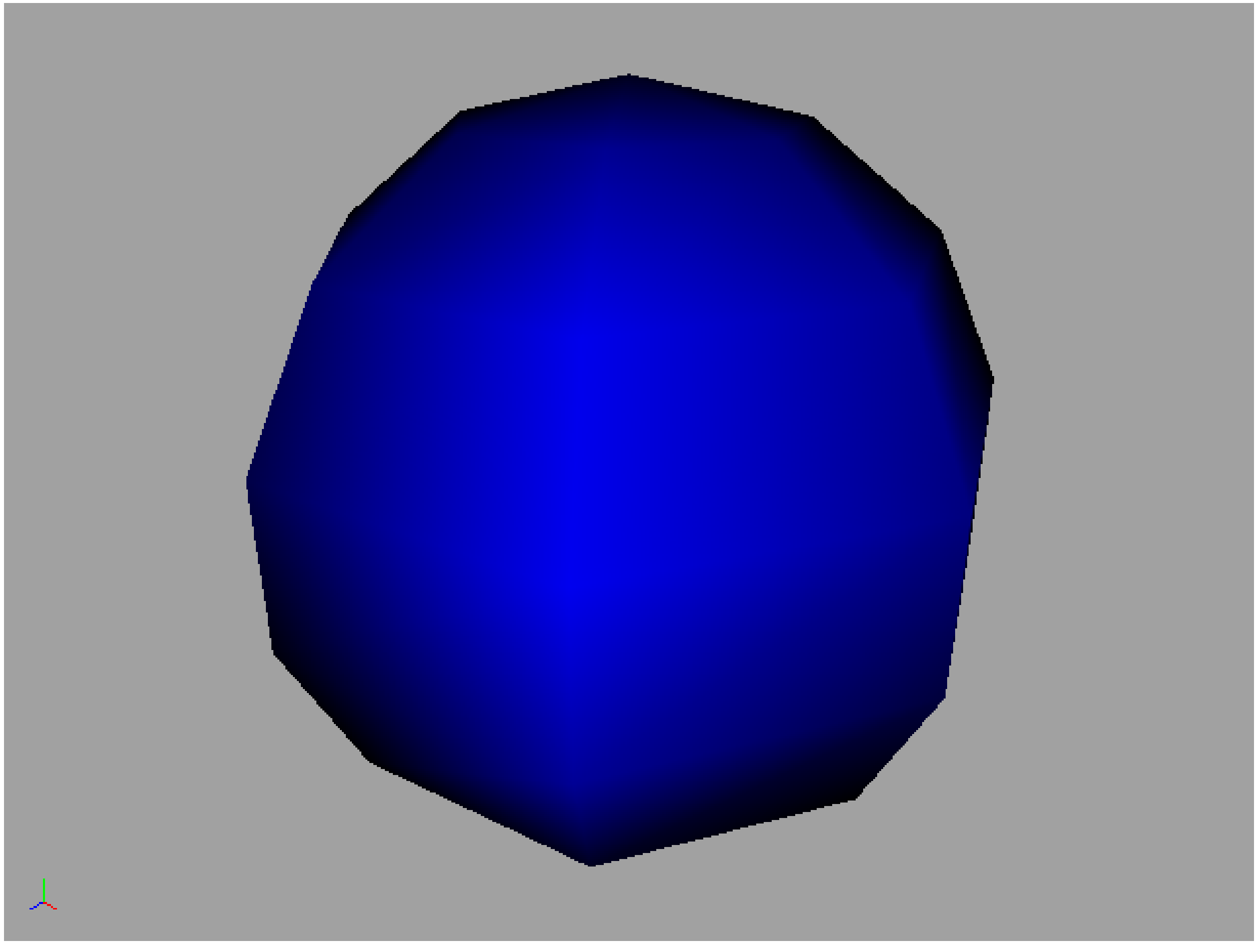
Smooth Shading (cont.)

- Each vertex is given a color
- Colors are interpolated between vertexes
- Vertex ordering may cause artifacts during interpolation
- Still no sense of depth

Lighting 101

- Simple lighting can be broken up into 3 separate components
- Ambient - Simulates secondary light i.e. light bouncing off walls
- Diffuse - Most closely models natural light coming from a light source
- Specular - Adds reflective highlights

Diffuse Phong Lighting



Diffuse Term

$$\text{diffuse} = (\max\{\mathbf{L} \cdot \mathbf{N}, 0\}) \times \text{diffuse}_{\text{light}} \times \text{diffuse}_{\text{material}}$$

- $\mathbf{L}$ = unit vector from the vertex to the light
- $\mathbf{N}$ = unit normal at the vertex
- $\text{diffuse}_{\text{light}}$ = diffuse color of the light
- $\text{diffuse}_{\text{material}}$ = diffuse color of the vertex

Ambient Term

- Global light needs to be accounted for
- No direction
- Should be relatively dim

$$\text{ambient} = \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}}$$

Attenuation

- Light should fall off with distance

$$\text{attenuation factor} = \frac{1}{k_c + k_l d + k_q d^2}$$

- d = distance from the vertex to the light
- k_c = constant attenuation constant
- k_l = linear attenuation constant
- k_q = quadratic attenuation constant

Putting It All Together

$$\text{color} = \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} +$$

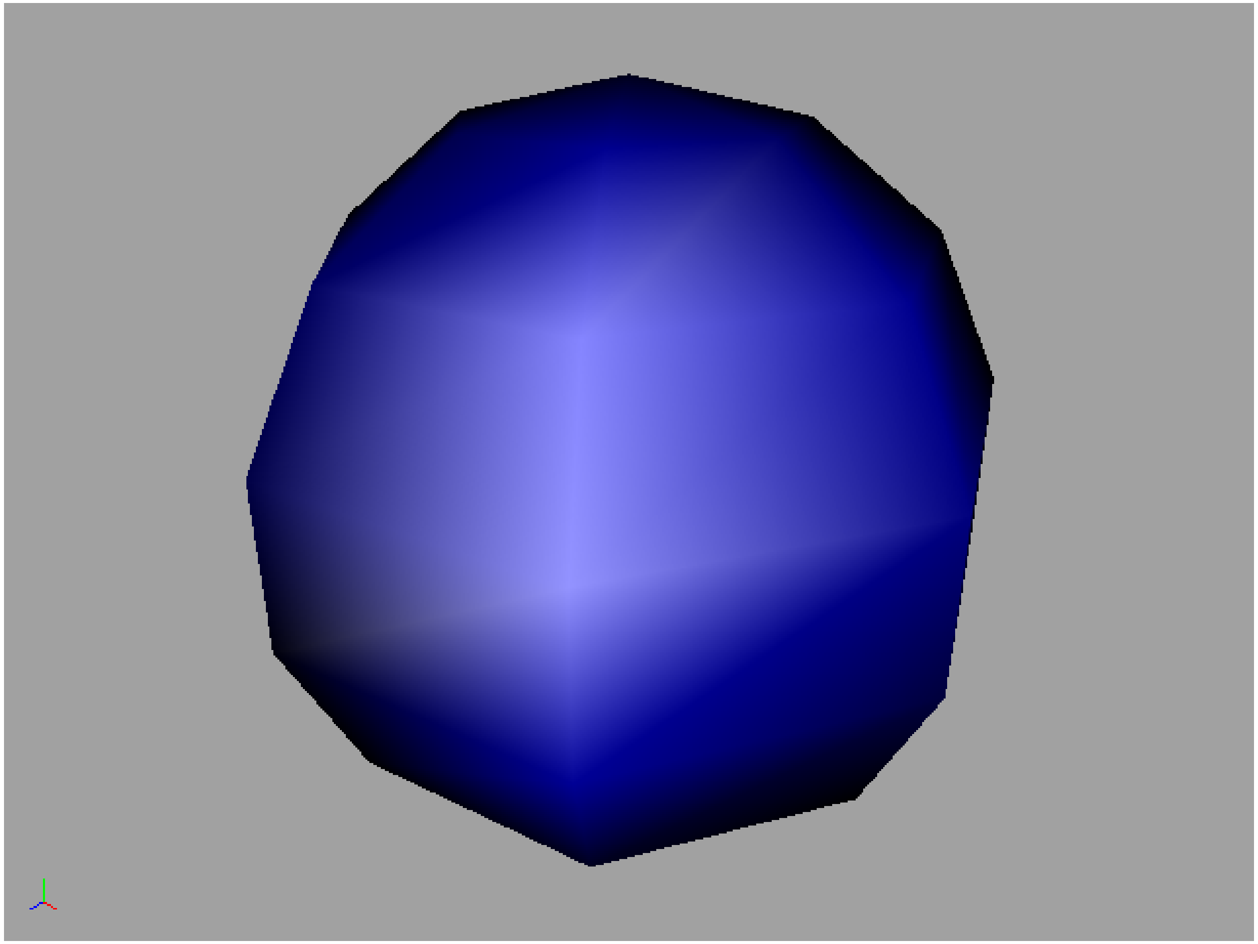
Putting It All Together

$$\begin{aligned}\text{color} &= \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} + \\ &\left(\frac{1}{k_c + k_l d + k_q d^2}\right) \times\end{aligned}$$

Putting It All Together

$$
\begin{aligned}
\text{color} &= \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} + \\
&\quad \left(\frac{1}{k_c + k_l d + k_q d^2}\right) \times \\
&\quad \left[(\max\{\mathbf{L} \cdot \mathbf{N}, 0\}) \times \text{diffuse}_{\text{light}} \times \text{diffuse}_{\text{material}}\right]
\end{aligned}
$$

Diffuse and Specular Lighting



Specular Term

$$\begin{aligned} \text{specular} &= (\max\{\mathbf{H} \cdot \mathbf{N}, 0\})^{\text{shininess}} \times \\ &\quad \text{specular}_{\text{light}} \times \text{specular}_{\text{material}} \end{aligned}$$

- If $\mathbf{L} \cdot \mathbf{N} = 0$ the specular term is 0
- $\mathbf{H}$ = the half vector

  a unit vector half way between the unit vector from the vertex towards the light

  and the unit vector from the vertex towards the eye position
- shininess = specular exponent

  controls the size of the specular highlight

Putting It All Together Again

$$\text{color} = \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} +$$

Putting It All Together Again

$$\text{color} = \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} + \left(\frac{1}{k_c + k_l d + k_q d^2}\right) \times$$

Putting It All Together Again

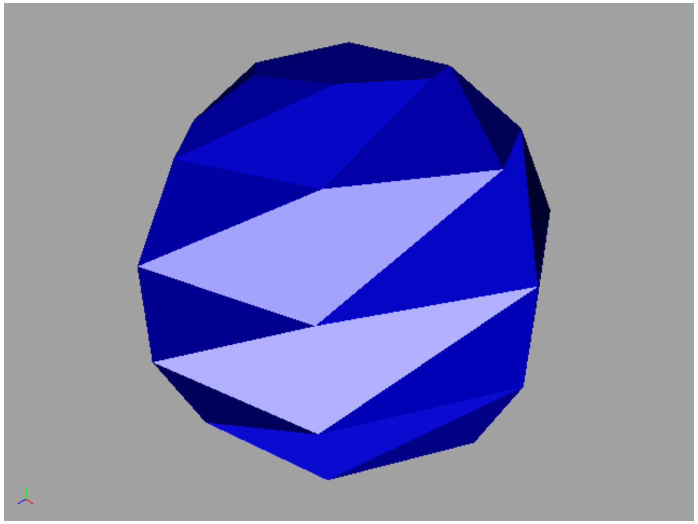
$$
\begin{aligned}
\text{color} \quad = \quad & \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} + \\
& \left(\frac{1}{k_c + k_l d + k_q d^2} \right) \times \\
& [(\max\{\mathbf{L} \cdot \mathbf{N}, 0\}) \times \text{diffuse}_{\text{light}} \times \text{diffuse}_{\text{material}} +
\end{aligned}
$$

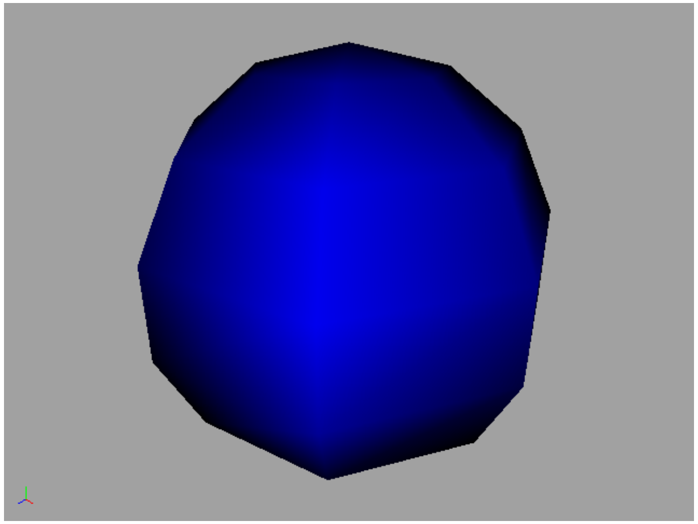
Putting It All Together Again

$$
\begin{aligned}
\text{color} \quad = \quad & \text{ambient}_{\text{light}} \times \text{ambient}_{\text{material}} + \\
& \left(\frac{1}{k_c + k_l d + k_q d^2} \right) \times \\
& [(\max\{\mathbf{L} \cdot \mathbf{N}, 0\}) \times \text{diffuse}_{\text{light}} \times \text{diffuse}_{\text{material}} + \\
& (\max\{\mathbf{H} \cdot \mathbf{N}, 0\})^{\text{shininess}} \times \text{specular}_{\text{light}} \times \text{specular}_{\text{material}}]
\end{aligned}
$$

Smooth/Flat Lighting

- Lighting can be computed either:
  - Per-polygon (flat shading)
  - Per-vertex (smooth shading)

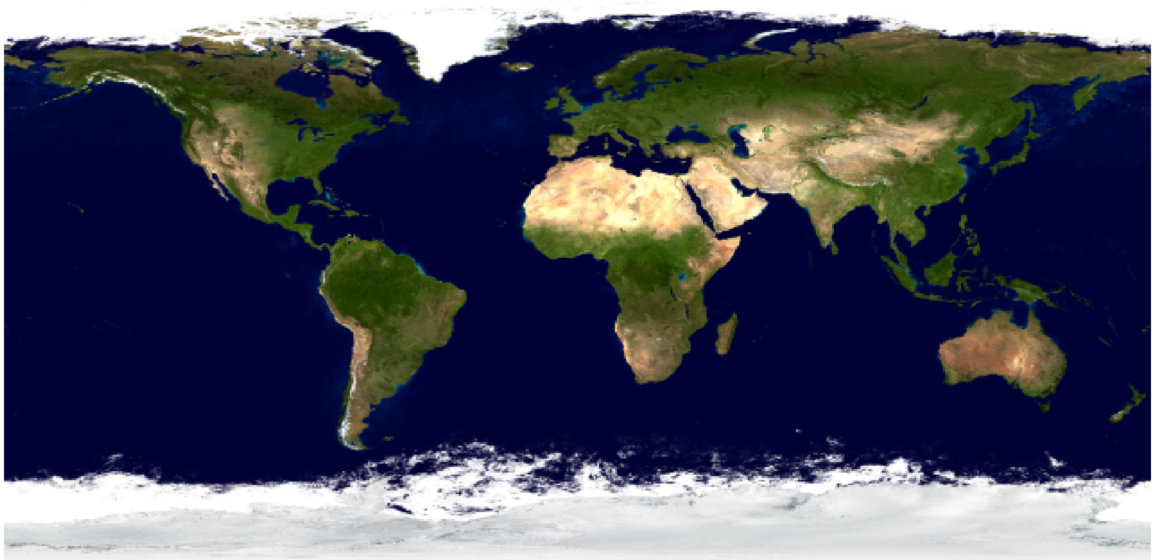




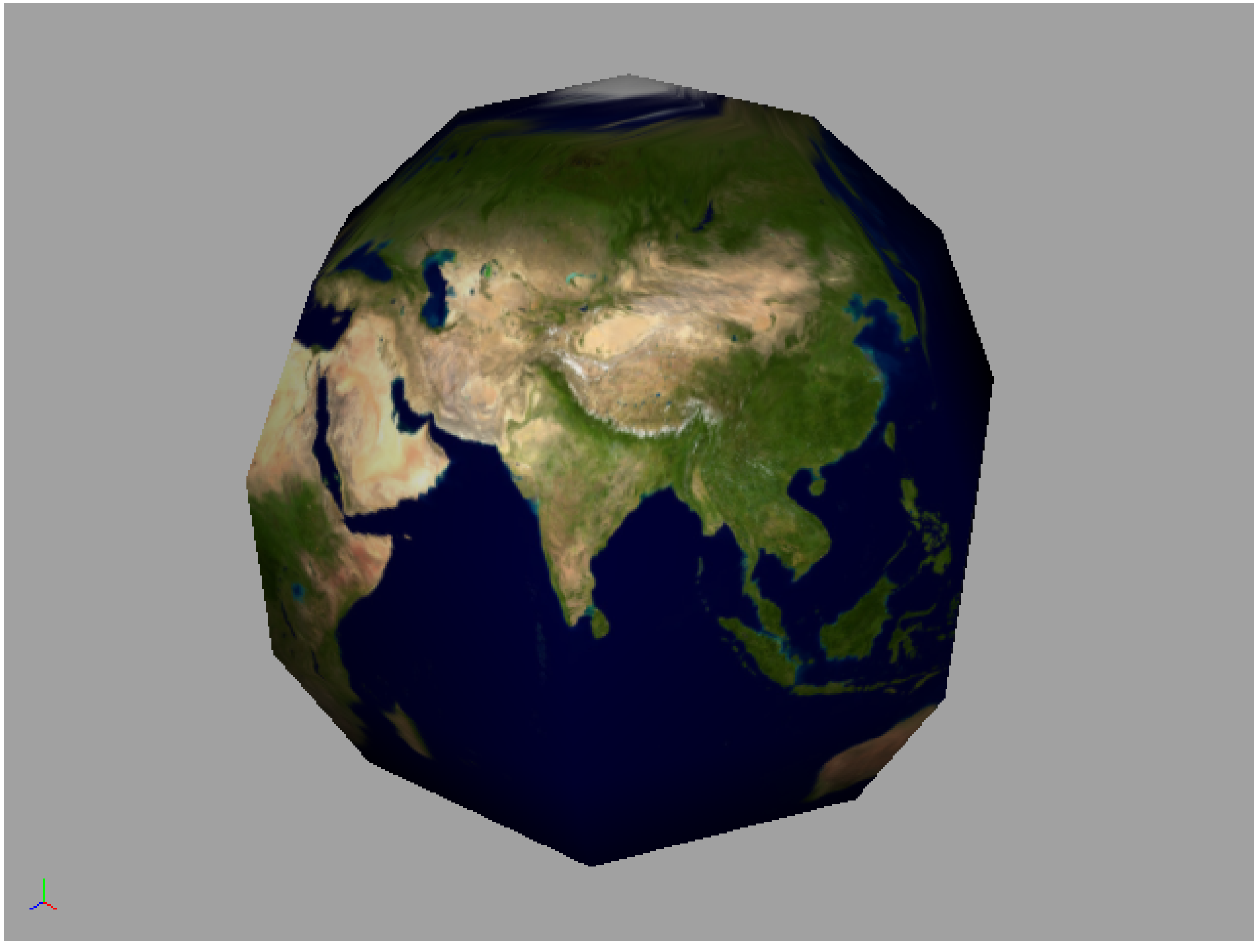
Demo

Texture Mapping

- Process of mapping a n-dimensional image onto a polygon
- Gives surface "texture"
- Adds detail without adding more polygons



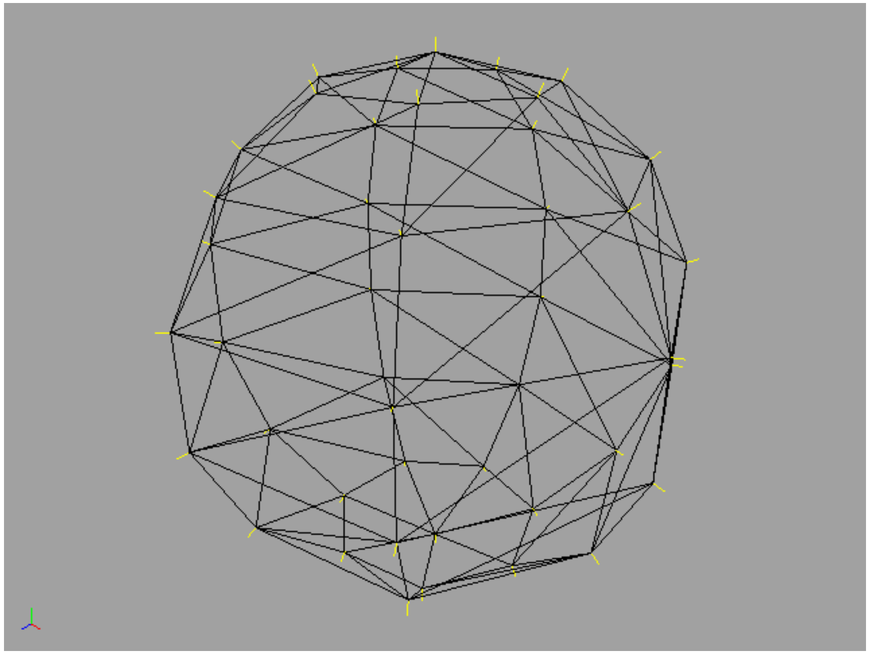
Texture Mapping Example

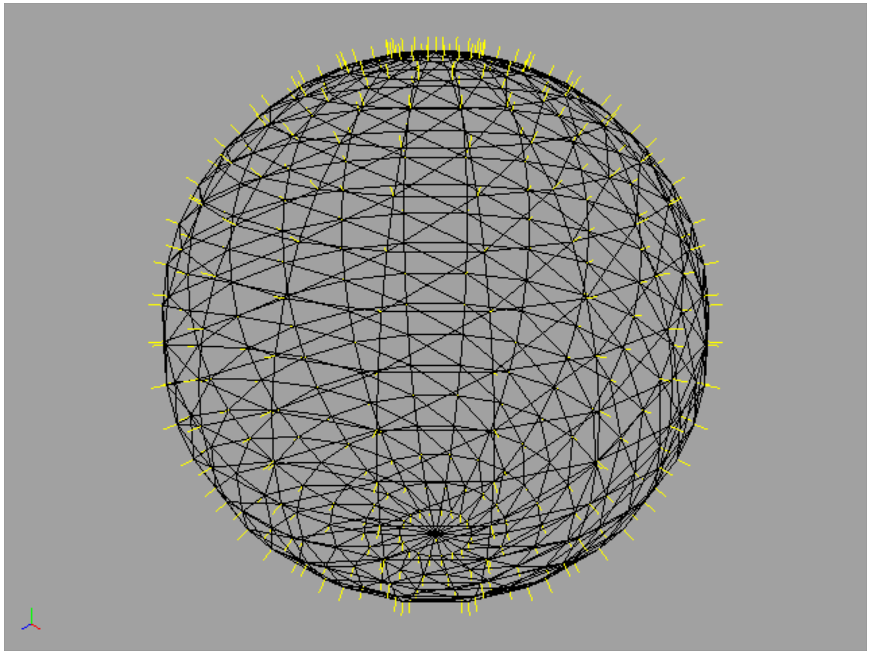


Shaping Up

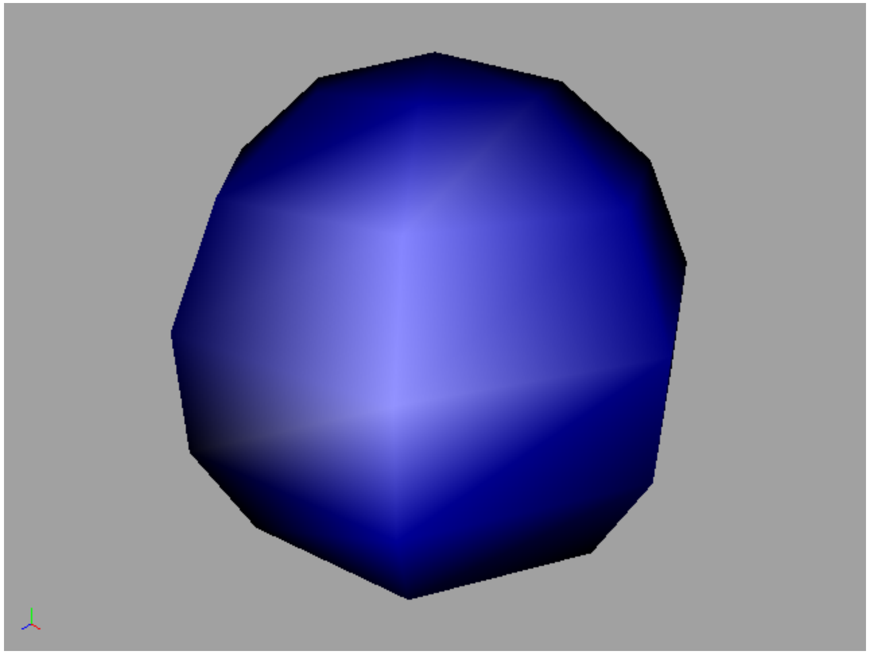
- Increasing the number of polygons in the object helps to further define the shape
- Adding vertexes makes per-vertex lighting computations more accurate
- Stabilizes specular highlights

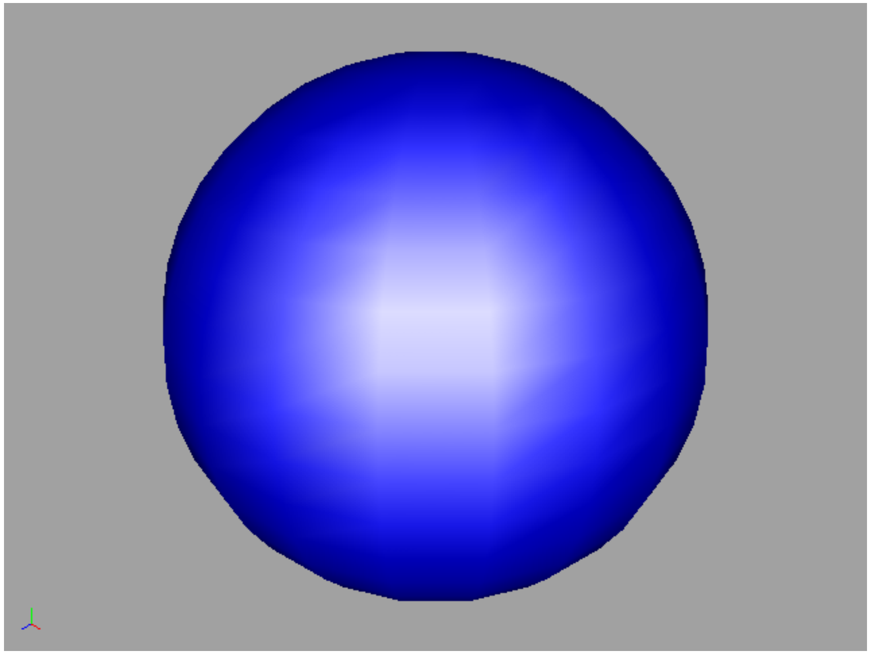
More Polygons



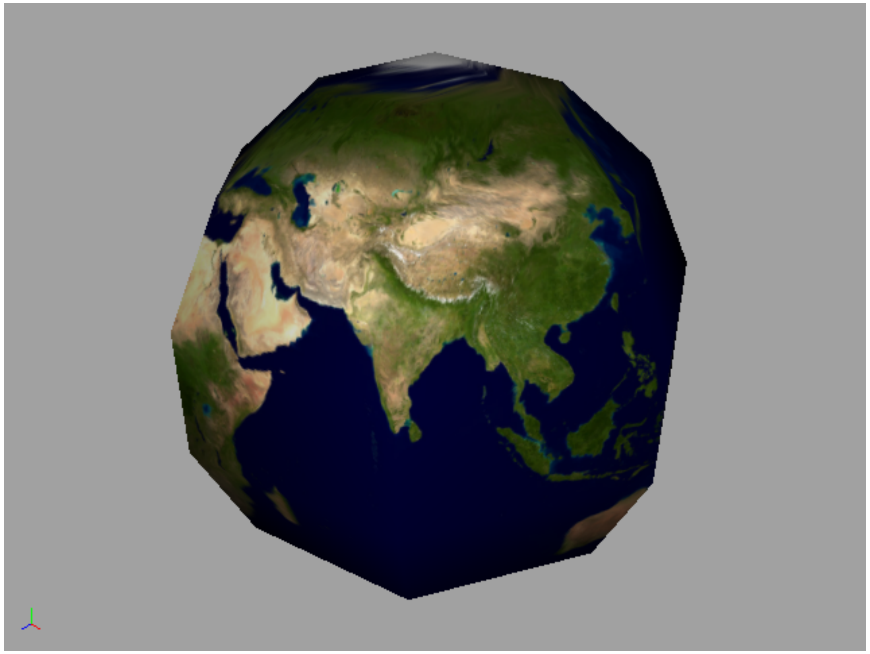


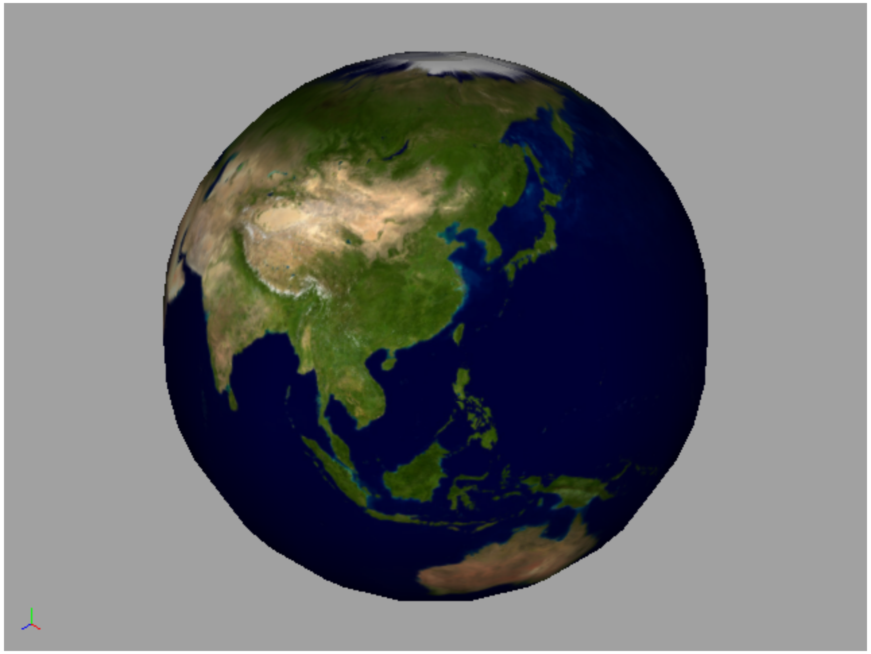
More Polygons





More Polygons





Demo

Basic OpenGL Lighting Summary

- Part of the OpenGL API

Basic OpenGL Lighting Summary

- Part of the OpenGL API
- Implemented in hardware

Basic OpenGL Lighting Summary

- Part of the OpenGL API
- Implemented in hardware
- Simulates plastic-like substance

Basic OpenGL Lighting Summary

- Part of the OpenGL API
- Implemented in hardware
- Simulates plastic-like substance
- Textures used for realism

Basic OpenGL Lighting Summary

- Part of the OpenGL API
- Implemented in hardware
- Simulates plastic-like substance
- Textures used for realism
- May require large polygon counts

Advanced Lighting Techniques

Goals

- We want our scene to be:
  - As close to photo-realistic as we can get
  - Interactive

Ideas

- We can add geometry:
  - HW can only handle a limited number of polygons
  - Memory may be limited

Ideas (cont.)

- We can use photo-realistic textures:
  - HW can only handle a limited number of polygons
  - Memory may be limited
  - Unless polygon count is high, scene may appear "flat"
  - Most textures encode shadow information

Solution

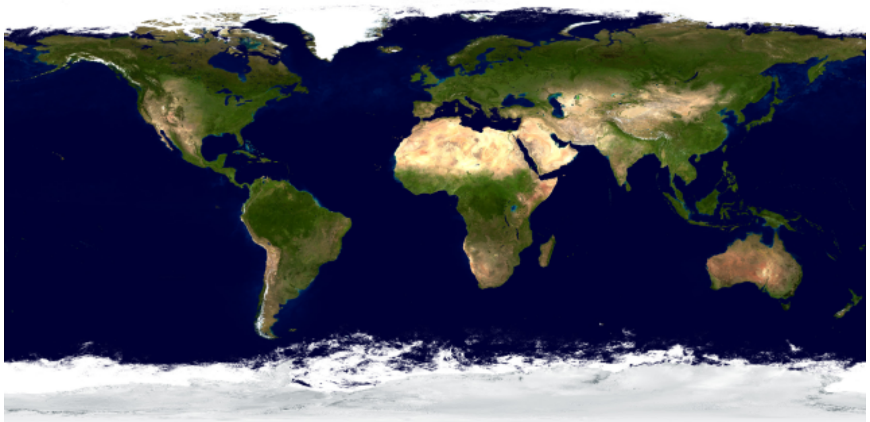
- Use a hybrid approach:
  - Use enough polygons to define the general shape of an object
  - Use texturing to define the objects details
- Most textures encode shadow information

Texture with Shadows

Multiple Textures

- Use multiple texture maps
- Each texture map represents a property of the surface:
  - Color Map - Defines the surface's color (without shadow information)
  - Bump Map - Greyscale image that defines the surface's height

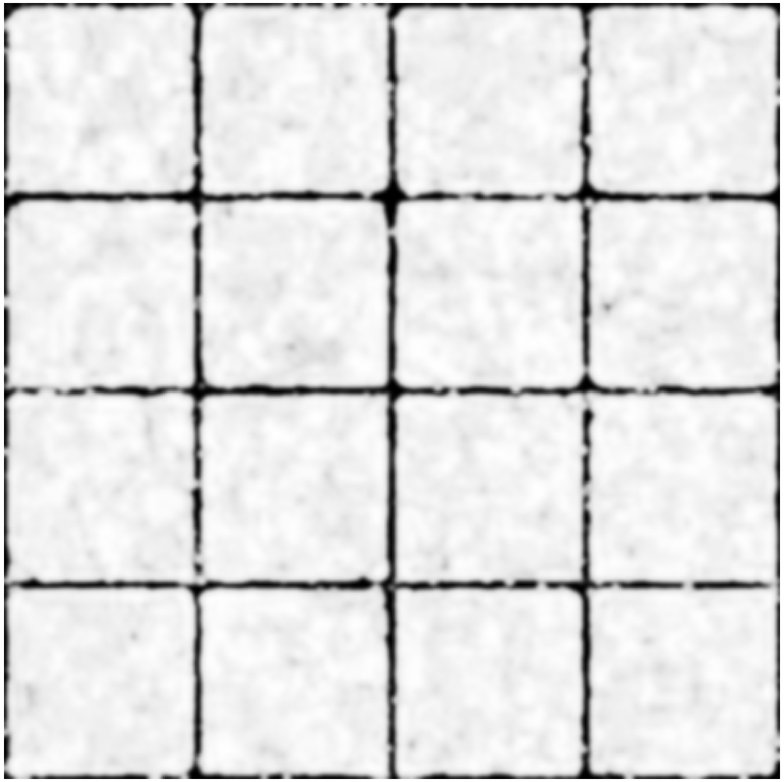
Color/Bump Map Example

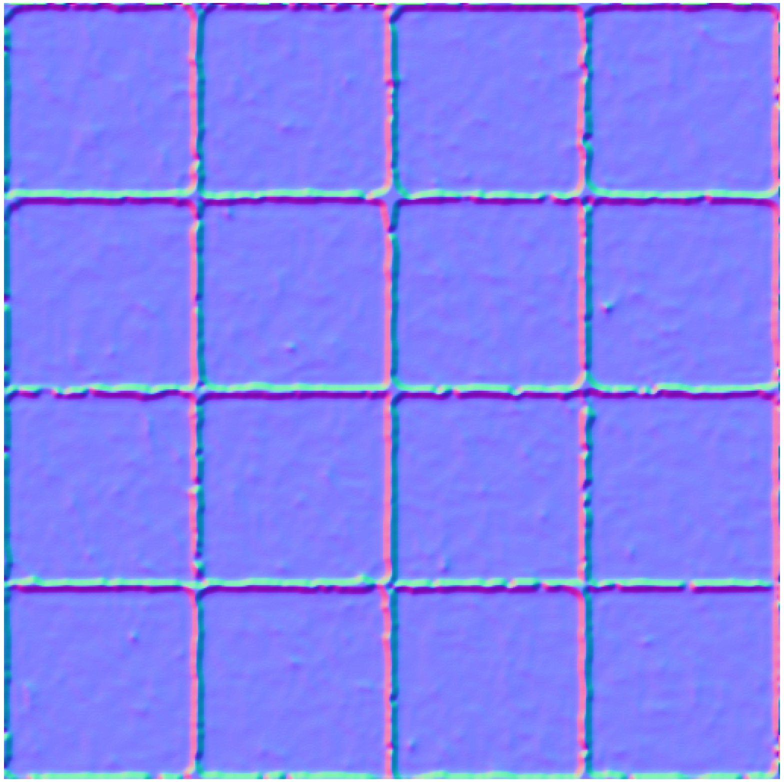


Key: Normal

- Specular and diffuse lighting terms both used the surface's normal
- We want to use the normal described by the height information in the bump map
- Generating normals from a bump map is straight-forward
- Bump Map $\rightarrow$ Normal Map

Normal Map Example

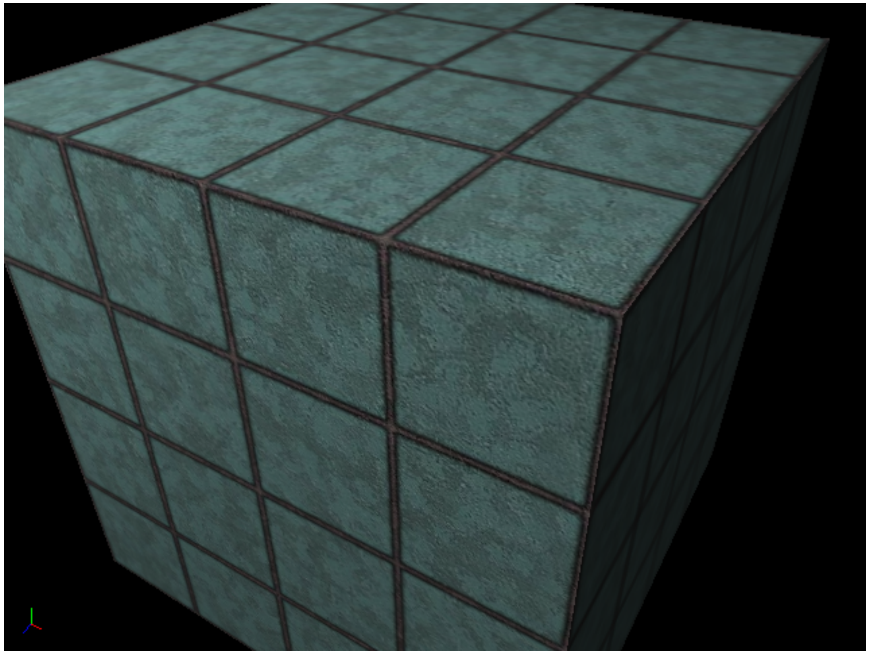


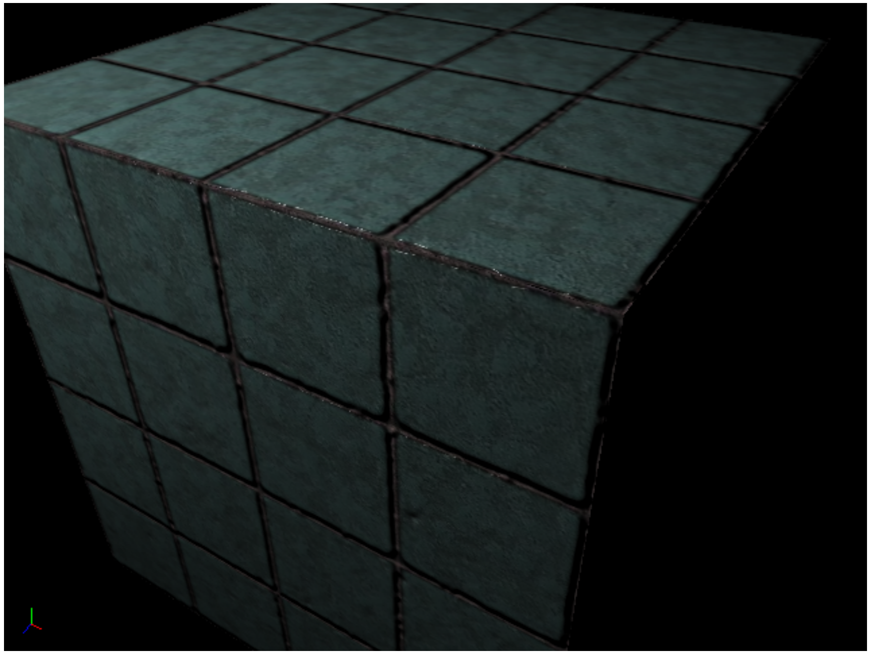


Normal Map

- RGB component of normal map represents the scaled XYZ components of the normal
- $map_i = \frac{normal_i}{2} + 0.5$
- Normal is defined in tangent space
  - Explains blue tint of normal map (Z component is always positive in tangent space)

Bump Mapping Example





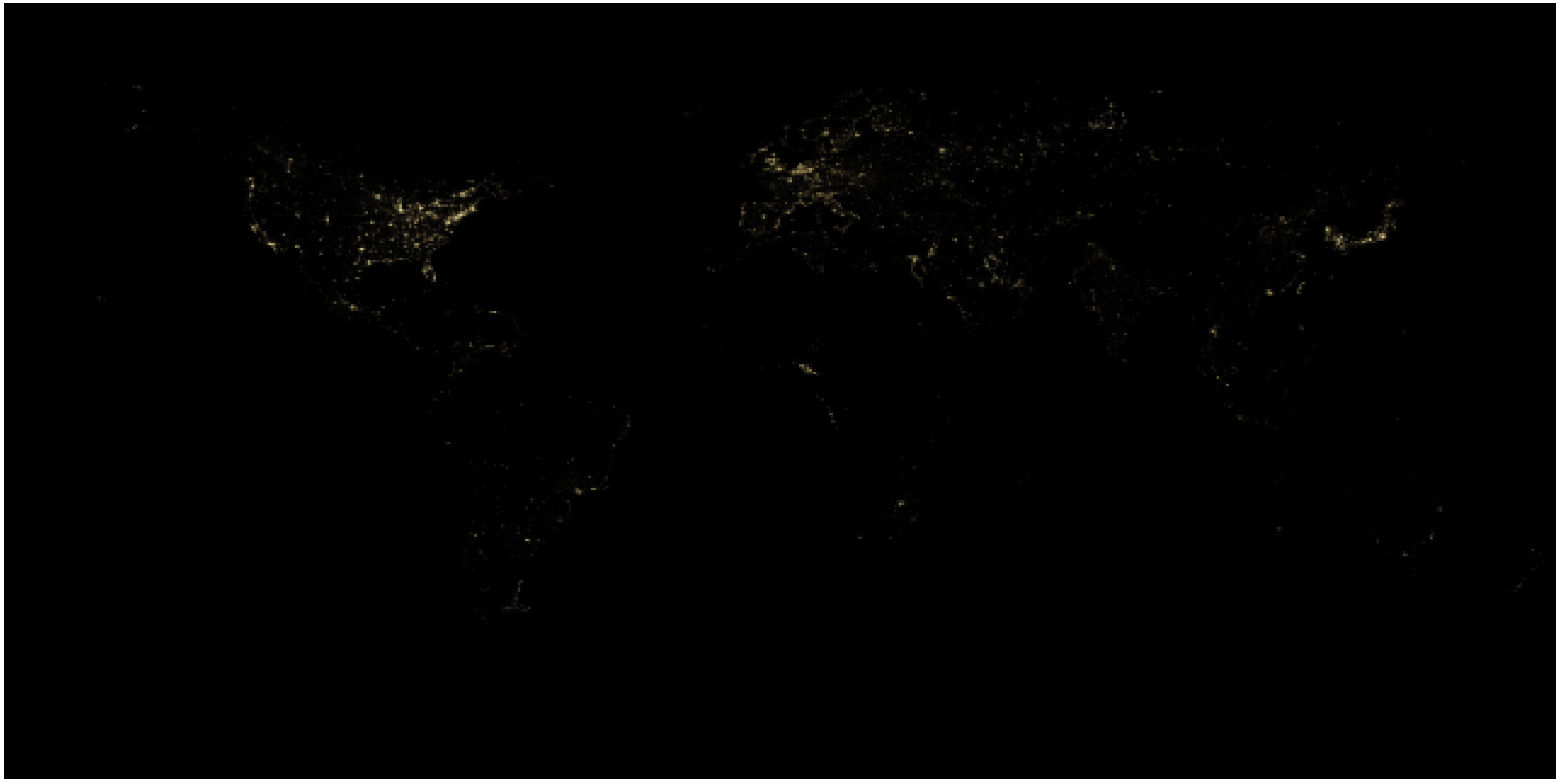
Demo

More Texture Maps

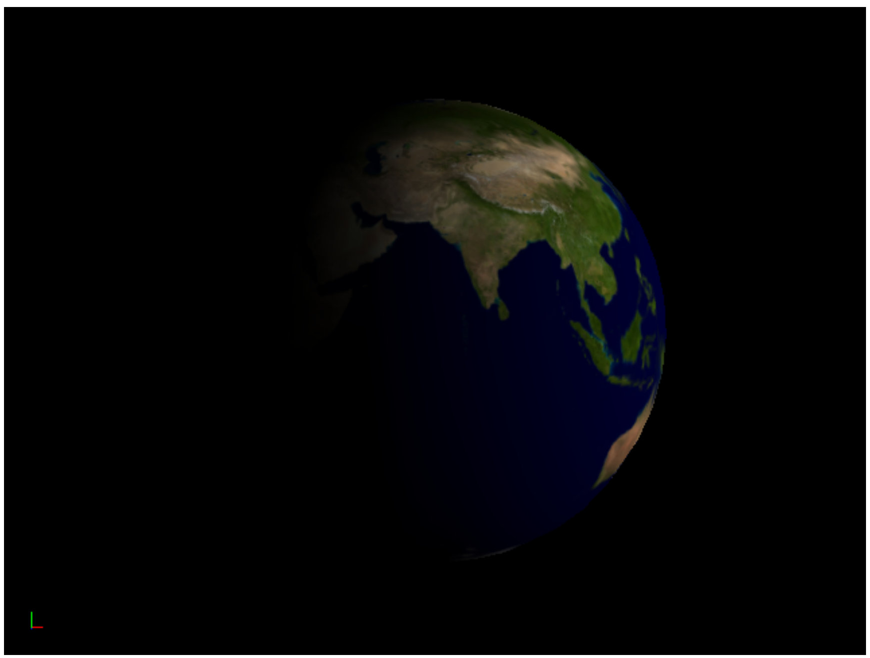
- We can add different types of texture maps to help us reach photorealism
- Specular Map - Modulates the specular component of the lighting equation
- Ambient Map - Adds ambient light to the surface

Specular Map Example

Ambient Map Example



Another Bump Mapping Example



Demo

Evolution of OpenGL

- Such demos were not possible 5 years ago
- OpenGL's graphics pipeline has evolved over the past 10 years:
  - State machine
  - Configurable (`GL_NV_register_combiners`)
  - Programmable with ASM like languages (`ARB_fragment/vertex_program`)
  - Programmable with high-level languages (`GLSlang`)

Evolution of OpenGL (cont.)

- The new programmable features of OpenGL allow the programmer to totally replace OpenGL's lighting equation

Summary

- Real time graphics have come a long way from flat shaded polygons
- We're getting closer to generating photo-realistic scenes in real time
- The techniques we've described will be what you'll see in real-time application for the next 5 years
- We've neglected to mention shadow generation

Plug

- Come to the graphics seminar to find out more
- Wednesday @ 2:30 Hidden somewhere in Riggs