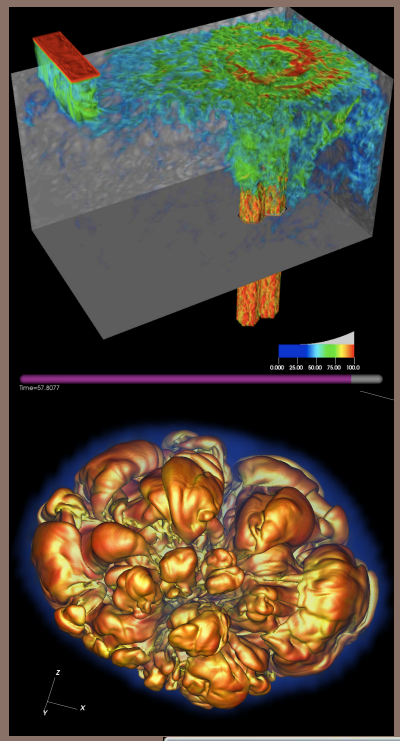
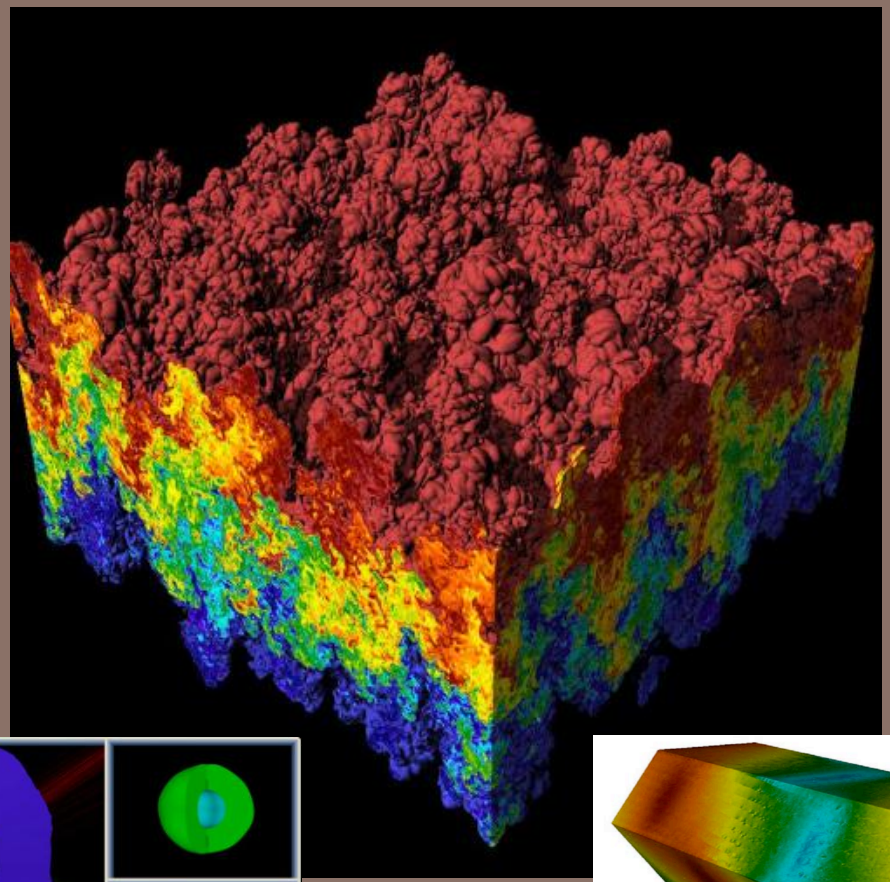
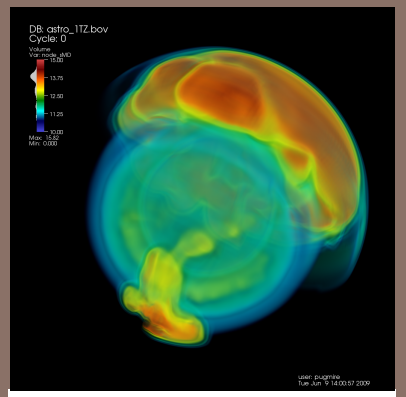
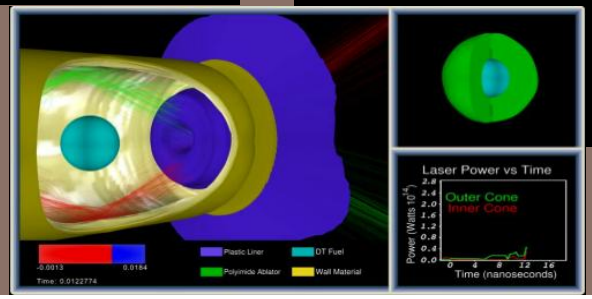
CIS 441/541: Introduction to Computer Graphics

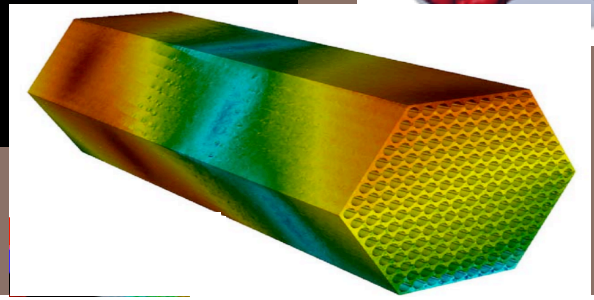
Lecture 8: Math Basics, Lighting Introduction & Phong Lighting









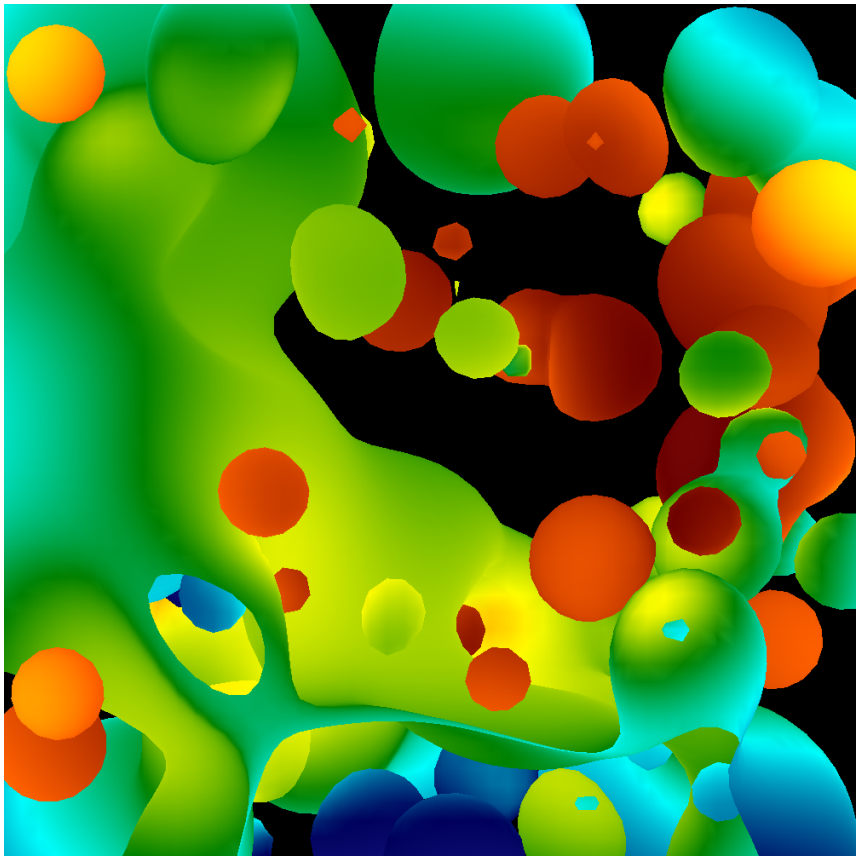


Oct. 11th, 2016

Hank Childs, University of Oregon

Project #1D (5%), Due Tues 10/18

- Goal: interpolation of color and zbuffer
- Extend your project1C code
- File proj1d_geometry.vtk available on web (1.4MB)
- File “reader1d.cxx” has code to read triangles from file.
- No Cmake, project1d.cxx



Color is now floating-point

- We will be interpolating colors, so please use floating point (0 → 1)
- Keep colors in floating point until you assign them to a pixel
- Fractional colors? → use ceil441…

Changes to data structures

```
class Triangle
{
  public:
      double X[3], Y[3], Z[3];
      double colors[3][3];
};
```

→ reader1d.cxx will not compile until you make these changes

Project 1D

- Everyone understand what to do?
- Reference:

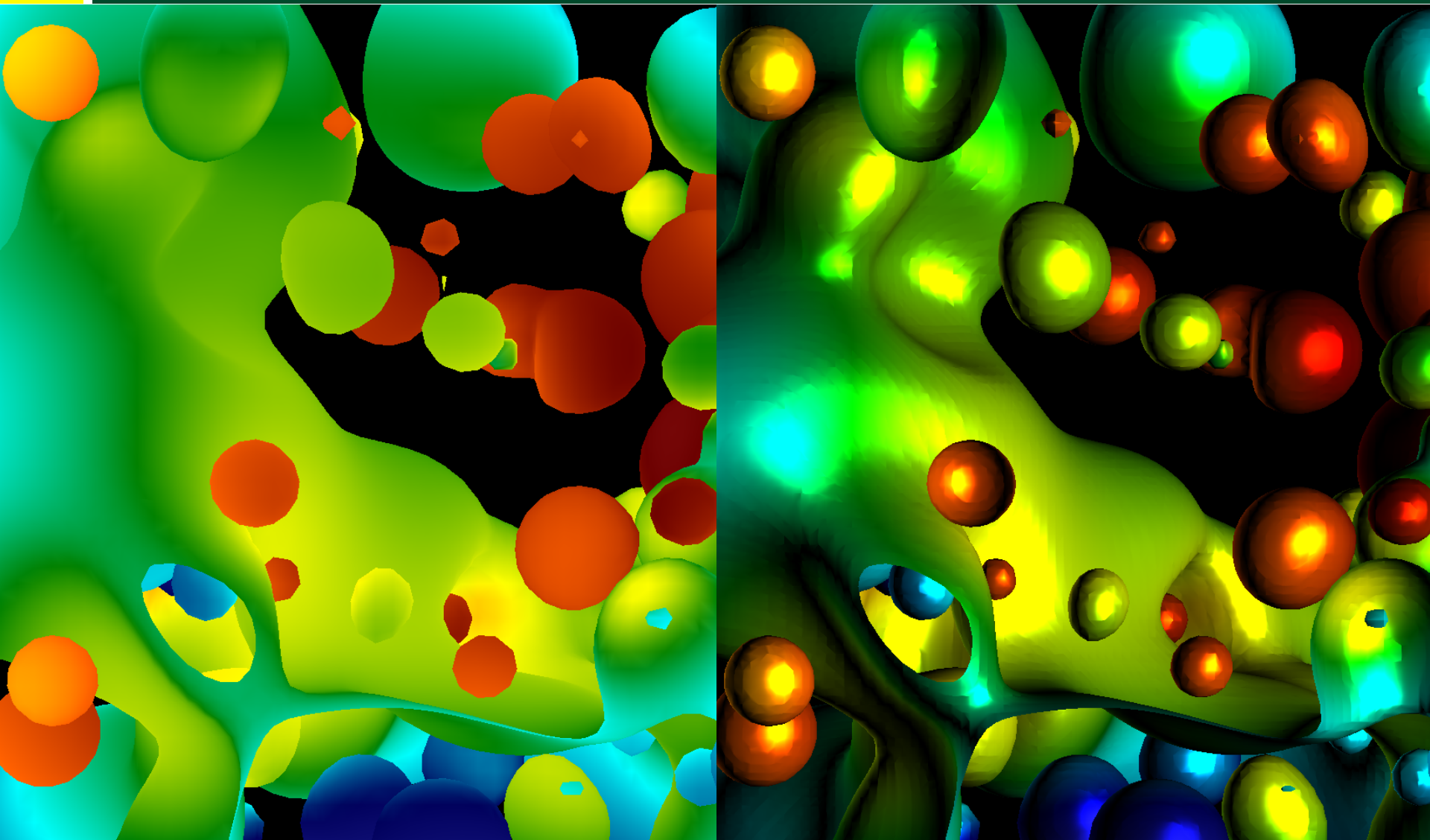
```
WORKING ON TRIANGLE 2


-- FLAT BOTTOM PART --
PRINTING THE TRIANGLE:

(551.02, 877.551, -0.953298)/(0.117997, 0.117997, 0.569589), (551.02, 864.703,
-0.938775)/(0.100532, 0.100532, 0.538104), (567.985, 864.703,
-0.941226)/(0.103373, 0.103373, 0.543226)

Scanline 865 with left intercept at 551.02, depth -0.939111, and color
0.100936/0.100936/0.538833, right intercept at 567.592, depth -0.941505, and
color 0.103712/0.103712/0.543836
Assigning pixel (552, 865) with color 25, 25, 137 and depth -0.939253
Assigning pixel (553, 865) with color 25, 25, 137 and depth -0.939397
Assigning pixel (554, 865) with color 25, 25, 137 and depth -0.939542
Assigning pixel (555, 865) with color 25, 25, 137 and depth -0.939686
Assigning pixel (556, 865) with color 25, 25, 137 and depth -0.939831
Assigning pixel (557, 865) with color 25, 25, 137 and depth -0.939975
Assigning pixel (558, 865) with color 26, 26, 137 and depth -0.94012
Assigning pixel (559, 865) with color 26, 26, 138 and depth -0.940264
Assigning pixel (560, 865) with color 26, 26, 138 and depth -0.940409
Assigning pixel (561, 865) with color 26, 26, 138 and depth -0.940553
Assigning pixel (562, 865) with color 26, 26, 138 and depth -0.940698
Assigning pixel (563, 865) with color 26, 26, 138 and depth -0.940842
Assigning pixel (564, 865) with color 26, 26, 138 and depth -0.940986
Assigning pixel (565, 865) with color 26, 26, 138 and depth -0.941131
Assigning pixel (566, 865) with color 26, 26, 138 and depth -0.941275
```

```
        unsigned char rgb[3];
        rgb[0] = (unsigned char) ceil441(255.0*(leftRGB[0] + proportion*
(rightRGB[0]-leftRGB[0])));
        rgb[1] = (unsigned char) ceil441(255.0*(leftRGB[1] + proportion*
(rightRGB[1]-leftRGB[1])));
        rgb[2] = (unsigned char) ceil441(255.0*(leftRGB[2] + proportion*
(rightRGB[2]-leftRGB[2])));
        double z = leftZ + proportion*(rightZ-leftZ);
        Assign(j, i, rgb, z);
```

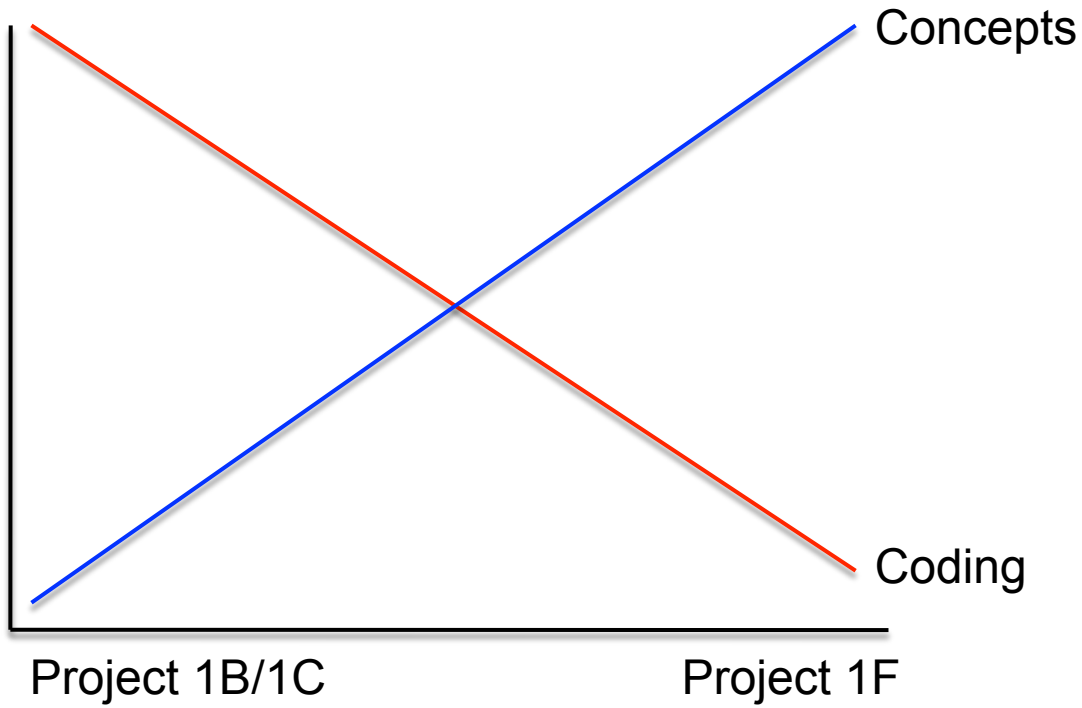
QuackCon Extension

- If you actively participate, then due October 20th
- Does not roll over to 1E

Our goal with 1E: add shading



My belief about Project #1X



Outline

- Math Basics
- Lighting Basics
- The Phong Model

Outline

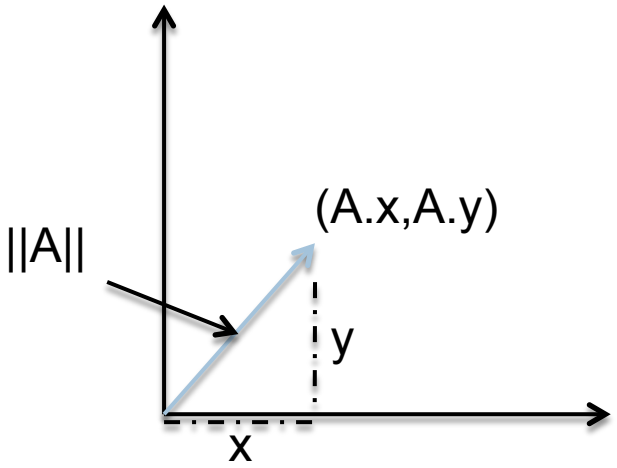
- Math Basics
- Lighting Basics
- The Phong Model

What is the norm of a vector?

- The norm of a vector is its length
  - Denoted with $||\cdot||$
- For a vector A = (A.x, A.y),

$$||A|| = \text{sqrt}(A.x*A.x+A.y*A.y)$$

- Physical interpretation:



- For 3D, $||A|| = \text{sqrt}(A.x*A.x+A.y*A.y+A.z*A.z)$

What does it means for a vector to be normalized?

- The vector A is normalized if $||A|| = 1$.
  - This is also called a unit vector.
- To obtain a normalized vector, take $A/||A||$
- Many of the operations we will discuss today will only work correctly with normalized vectors.

What is the normal of a triangle?

- A triangle coincides with a flat plane.
- A triangle's normal is the vector perpendicular to that plane.
- If a triangle is on plane $= Ax+By+Cz = D$,

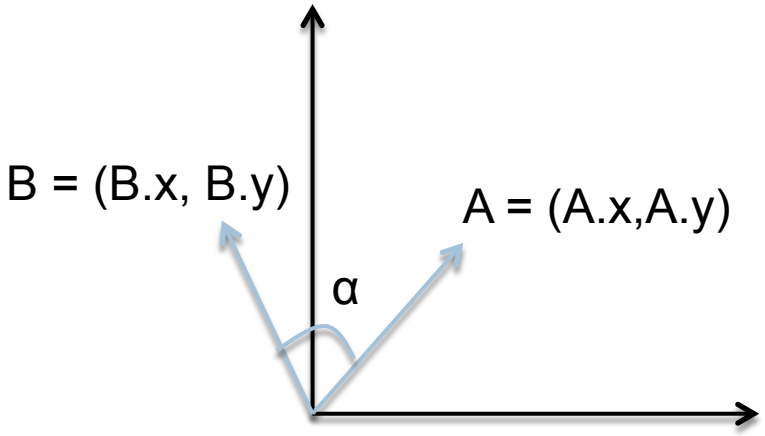
  then the triangle's normal is (A, B, C)

Norm, Normal, Normalize, Oh My!

- Norm: the length of a vector (||A||)
- Normal: a perpendicular vector to a plane coincident with geometry
- Normalize: the operation to create a vector with length 1 (A/||A||)
- All 3 are important for today's lecture

What is a dot product?

- $A \cdot B = A.x*B.x + A.y*B.y$
  - (or $A.x*B.x + A.y*B.y + A.z*B.z$)
- Physical interpretation:
  - $A \cdot B = \cos(\alpha)*(||A||*||B||)$



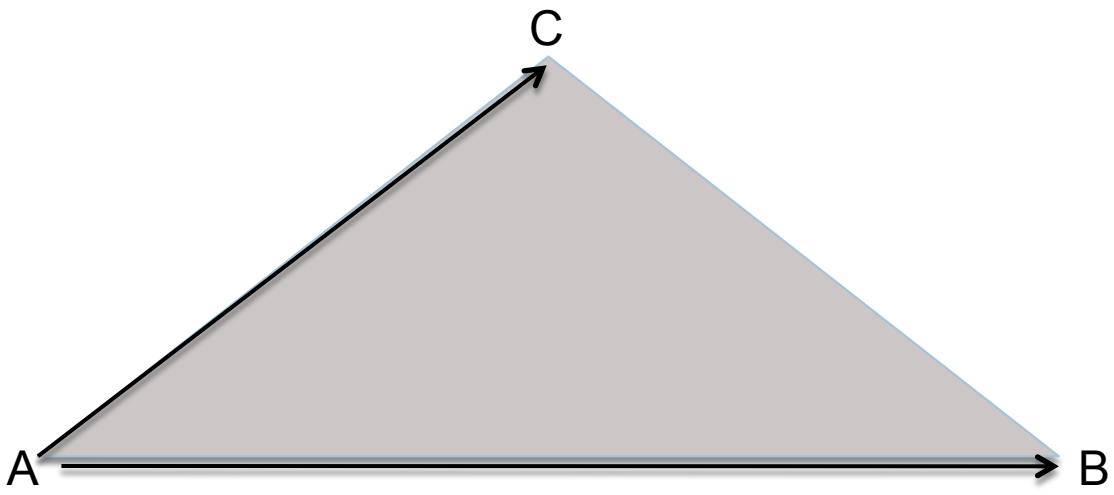
What is the cross product?

- AxB = (A.y*B.z - A.z*B.y,
  B.x*A.z - A.x*B.z,
  A.x*B.y - A.y*B.x)

- What is the physical interpretation of a cross product?
  - Finds a vector perpendicular to both A and B.

Easy Way to Calculate Normal For a Triangle

- Normal = (C-A)x(B-A)



Important:
(C-A)x(B-A) != (B-A)x(C-A)
… we'll worry about this later

Shading

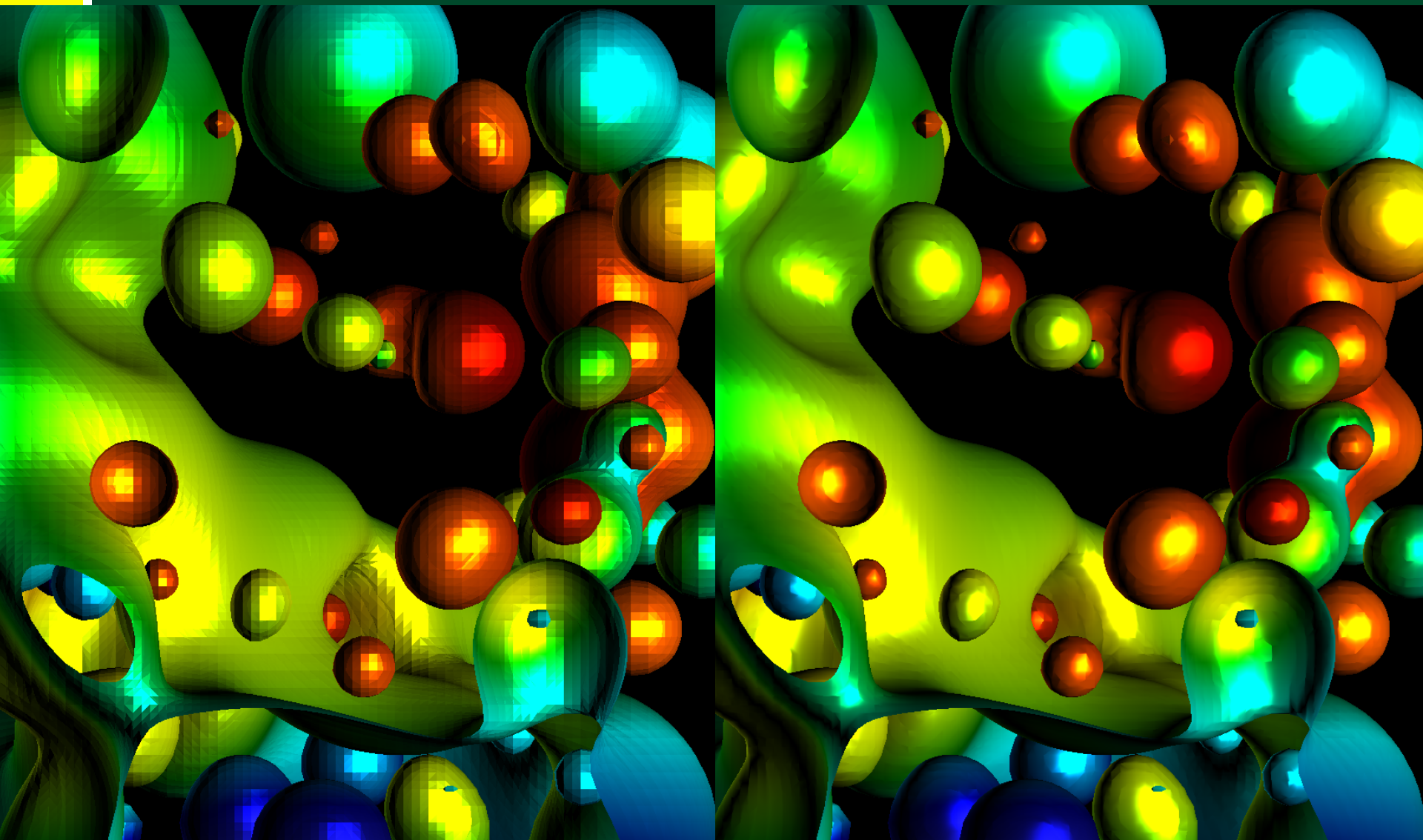
- Shading != Shadows
  - Shading:
    - brighter where light hits square
    - darker where light is tangent
  - Shadow: one object obscures light from getting to another object
- Shading & Normals
  - Normals – orientation of geometry with respect to light source
  - Key ingredient in recipe for shading
    - (so we will think about normals first and then learn how they are useful for shading after)

Lighting and Normals

- Two ways to treat normals:
  - Constant over a triangle
  - Varying over a triangle

- Constant over a triangle ←→ flat shading
- Varying over a triangle ←→ smooth shading

Flat vs Smooth Shading



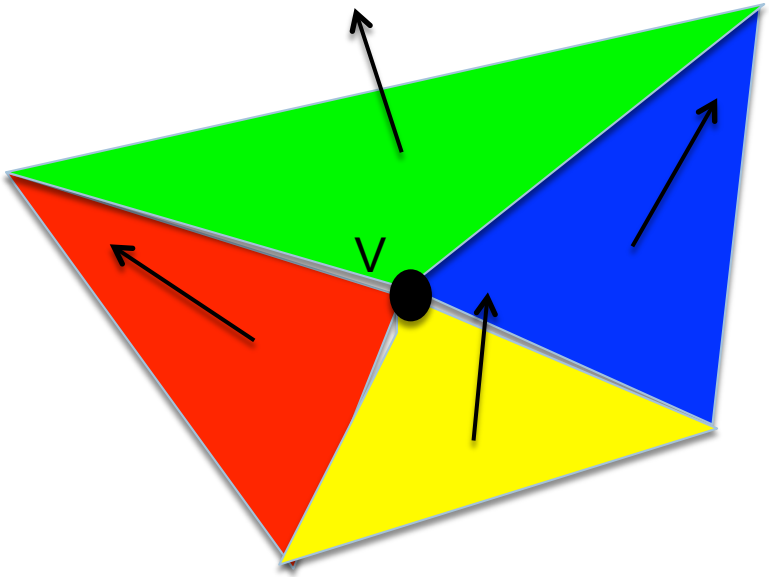
Lighting and Normals

- Two ways to treat normals:
  - Constant over a triangle
  - Varying over a triangle

- Constant over a triangle ←→ flat shading
  - Take (C-A)x(B-A) as normal over whole triangle
- Varying over a triangle ←→ smooth shading
  - Calculate normal at vertex, then use linear interpolation
    - How do you calculate normal at a vertex?
    - How do you linearly interpolate normals?

Vertex Normals

- Algorithm:
  - For vertex V,
    - Find all triangles T_i incident to V
    - Normal(V) = {0,0,0}
    - NumIncident = 0
    - For each T_i,
      - calculate Normal(T_i)
      - Normal(V) += Normal(T_i)
      - NumIncident++
    - Normal(V) /= NumIncident

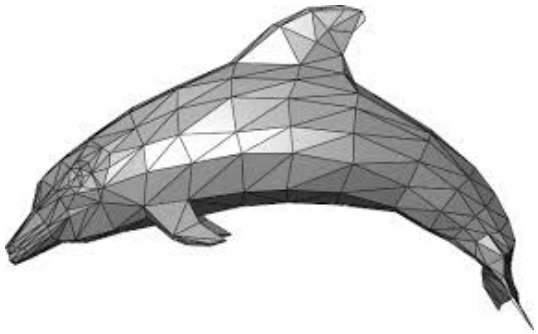


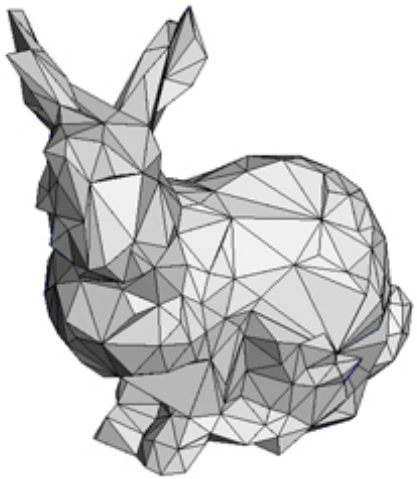
N(V) = (N(T1)+N(T2)+N(T3)+N(T4)) / 4

- Note: our data structures don't allow for "Find all triangles T_i incident to V" very easily.
  - Vertex normals are precalculated for 1E

Surfaces

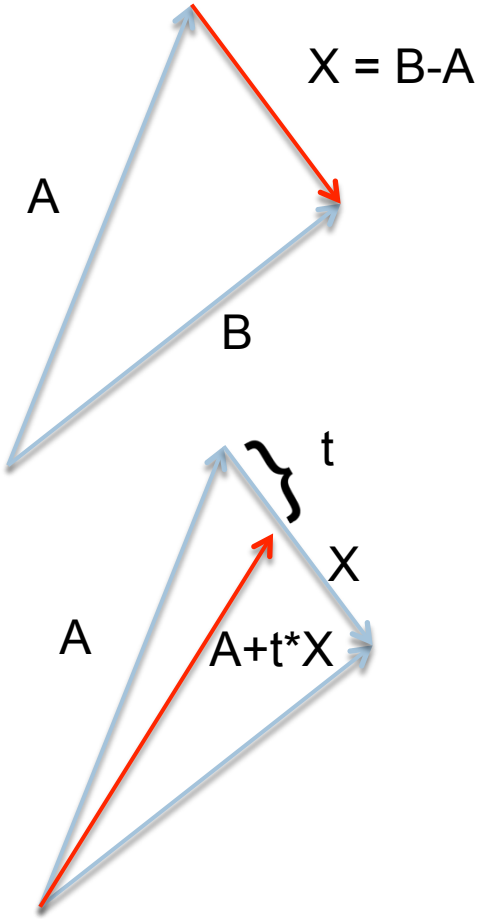
- For the most part, input surfaces will be connected in a “mesh” of triangles
- So it is meaningful to consider a vertex and the triangles surrounding that vertex



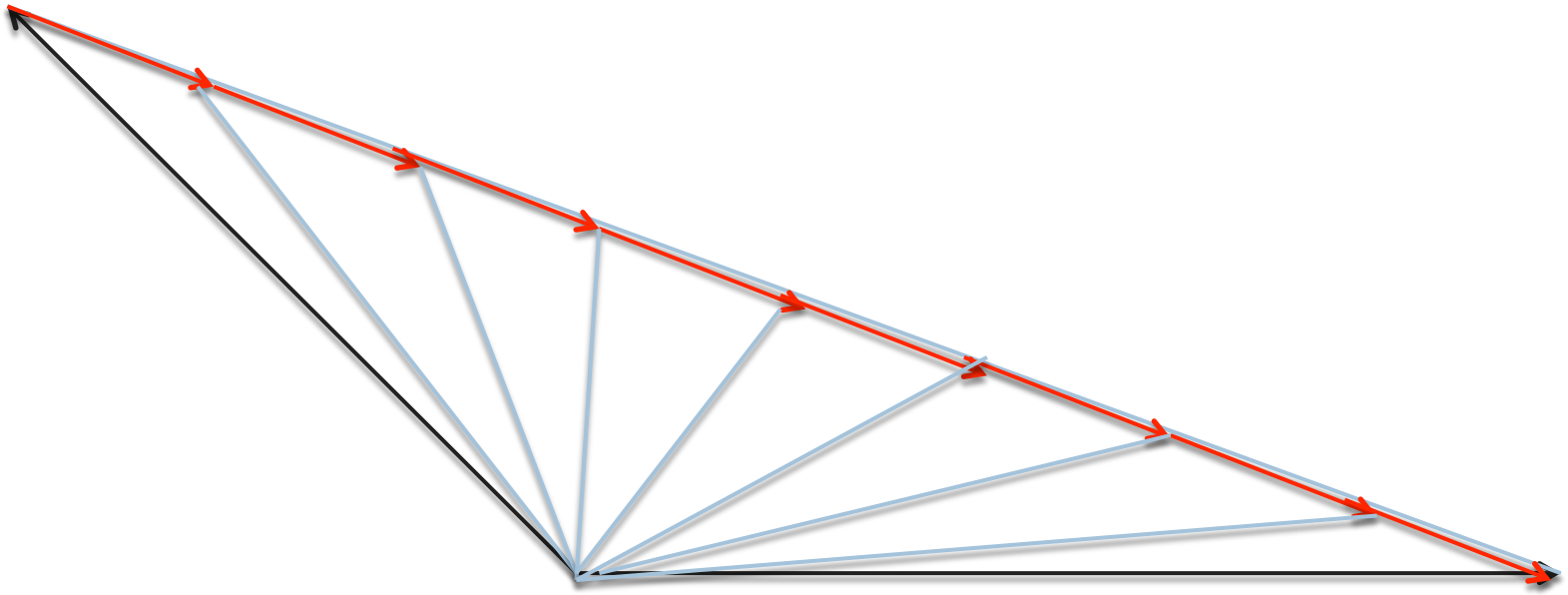


LERPing vectors

- LERP = Linear Interpolate
- Goal: interpolate vector between A and B.
- Consider vector X, where X = B-A
- Back to normal LERP:
  - A + t*(B-A) = A+t*X
- You will need to LERP vectors for 1E



Masado's Comment (Approximately)



Even steps in t do not lead to even steps in angle
Also, resulting vector is likely not a normal

What you need to understand:
only that we will LERP vectors on a component by component basis
AND you should normalize them after you LERP

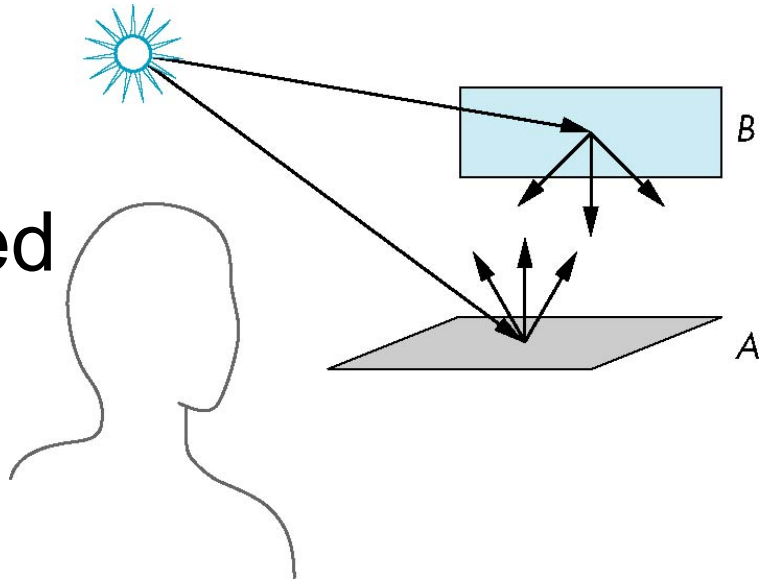
Outline

- Math Basics
- Lighting Basics
- The Phong Model

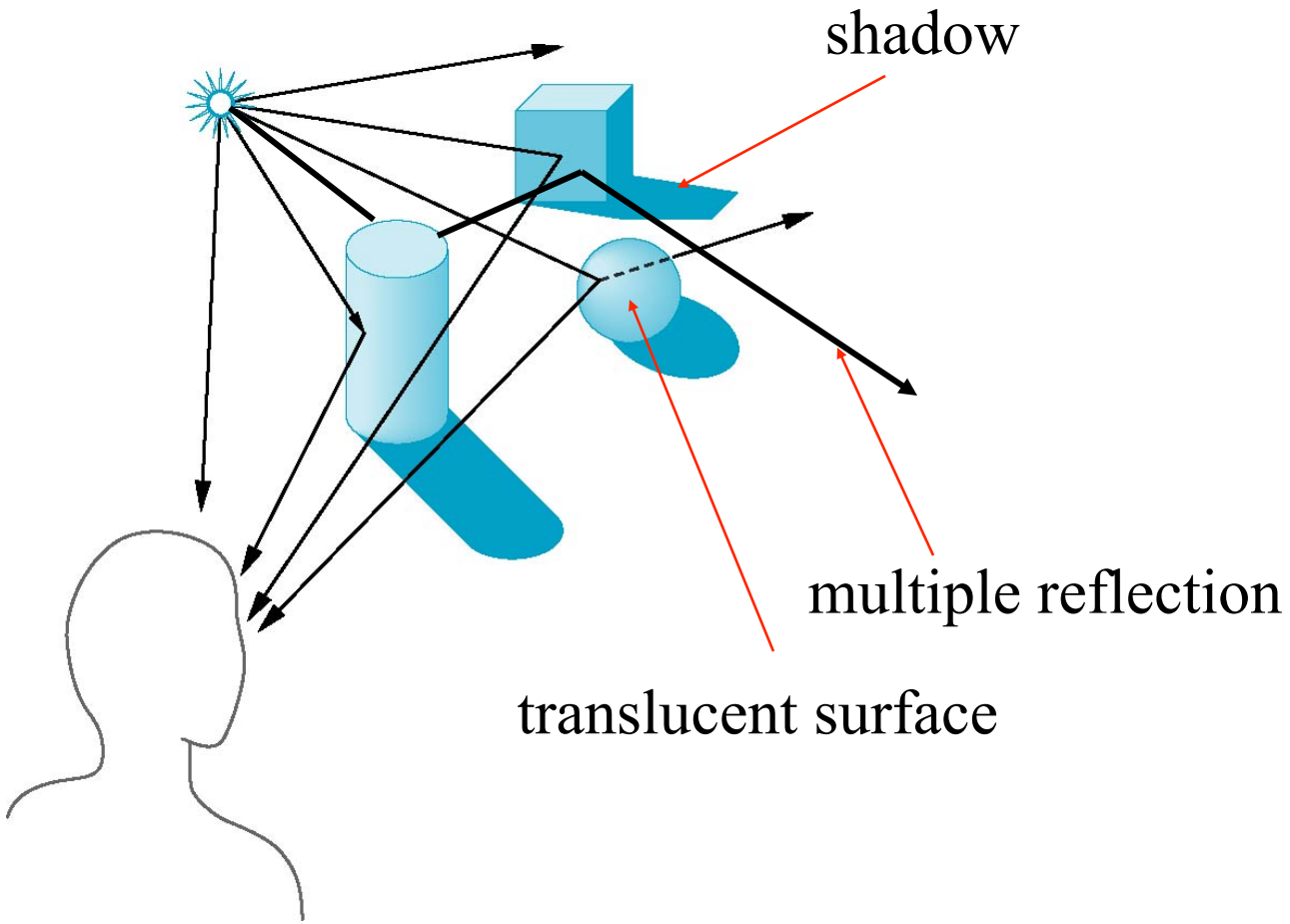
Scattering

- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A

  and so on



Global Effects



Local vs Global Rendering

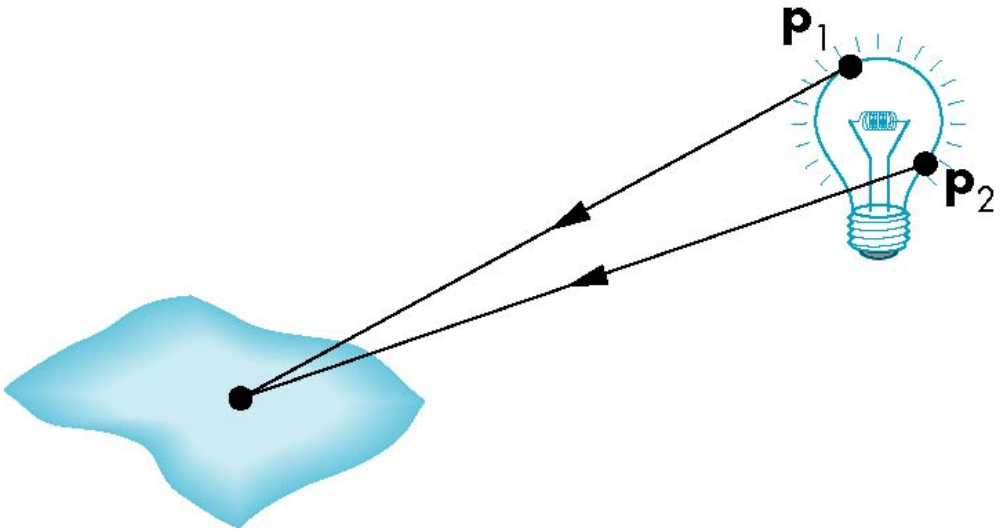
- Correct shading requires a global calculation involving all objects and light sources
  - Incompatible with model which shades each polygon independently (local rendering)
- However, in computer graphics, especially real time graphics, we are happy if things "look right"
  - Many techniques exist for approximating global effects

Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The amount reflected determines the color and brightness of the object
  - A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Light Sources

General light sources are difficult to work with because we must integrate light coming from all points on the source



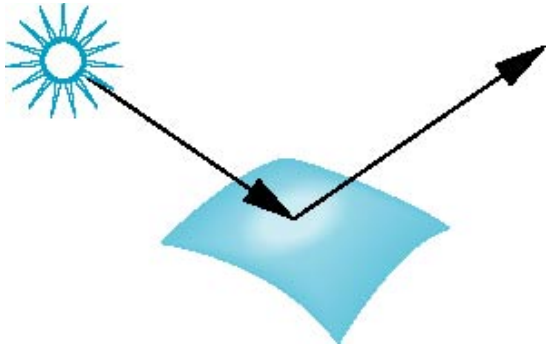
Simple Light Sources

- Point source
  - Model with position and color
  - Distant source = infinite distance away (parallel)
- Spotlight
  - Restrict light from ideal point source
- (We will do point sources for 1E … and this class)

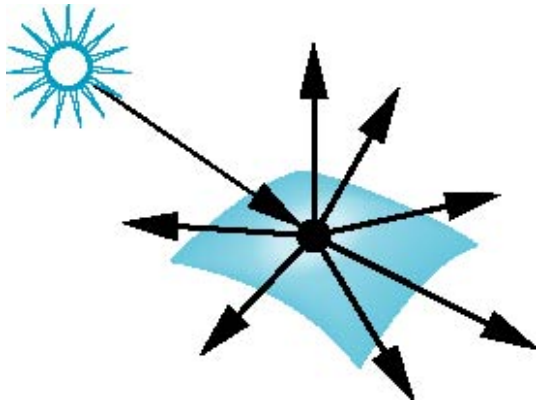
The University of New Mexico

Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A very rough surface scatters light in all directions



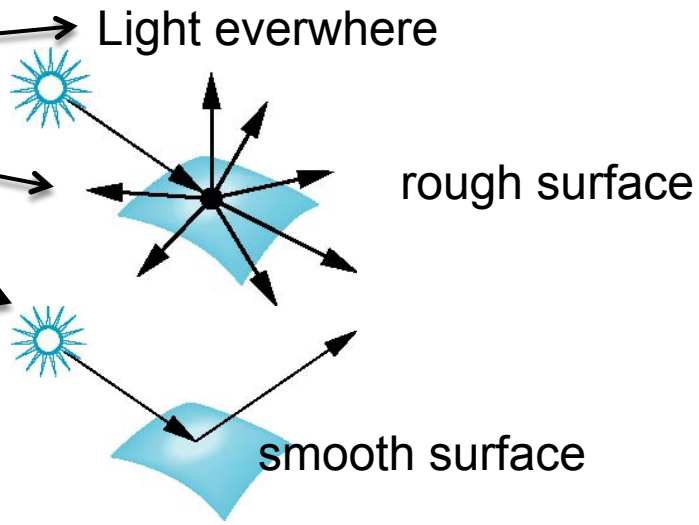
smooth surface



rough surface

Shading

- Our goal:
  - For each pixel, calculate a shading factor
  - Shading factor typically between 0 and 1, but sometimes >1
    - Shading >1 makes a surface more white
- 3 types of lighting to consider:
  - Ambient
  - Diffuse
  - Specular



Our game plan:
Calculate all 3 and combine them.

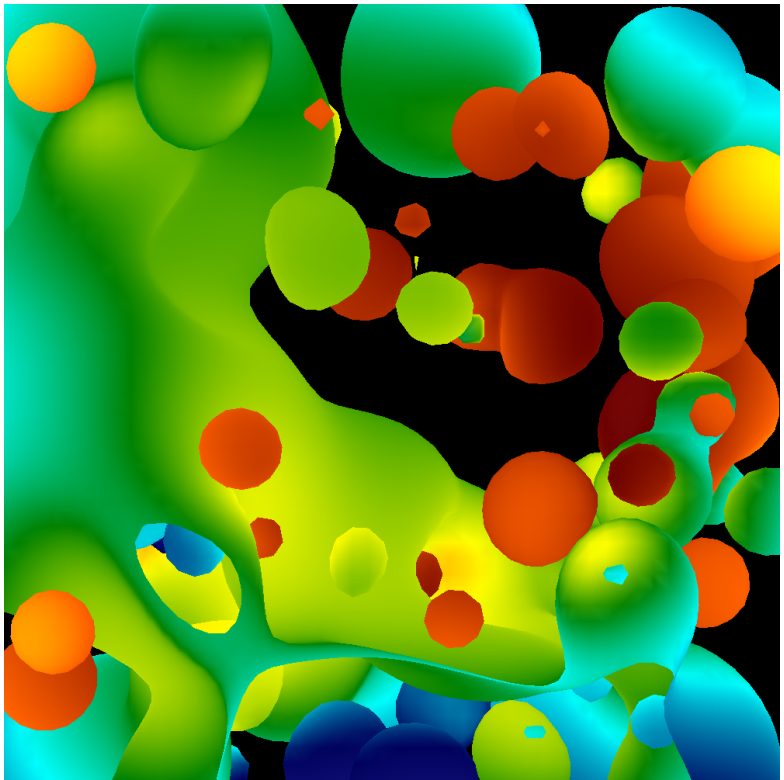
How to handle shading values greater than 1?

- Color at pixel = (1.0, 0.4, 0.8)
- Shading value = 0.5
  - Easy!
  - Color = (0.5, 0.2, 0.4)→ (128, 52, 103)
- Shading value = 2.0
  - Color = (1.0, 0.8, 1.0) → (255, 204, 255)
- Color_R = 255*min(1, R*shading_value)
- This is how bright lights makes things whiter and whiter.
  - But it won't put in colors that aren't there.

Ambient Lighting

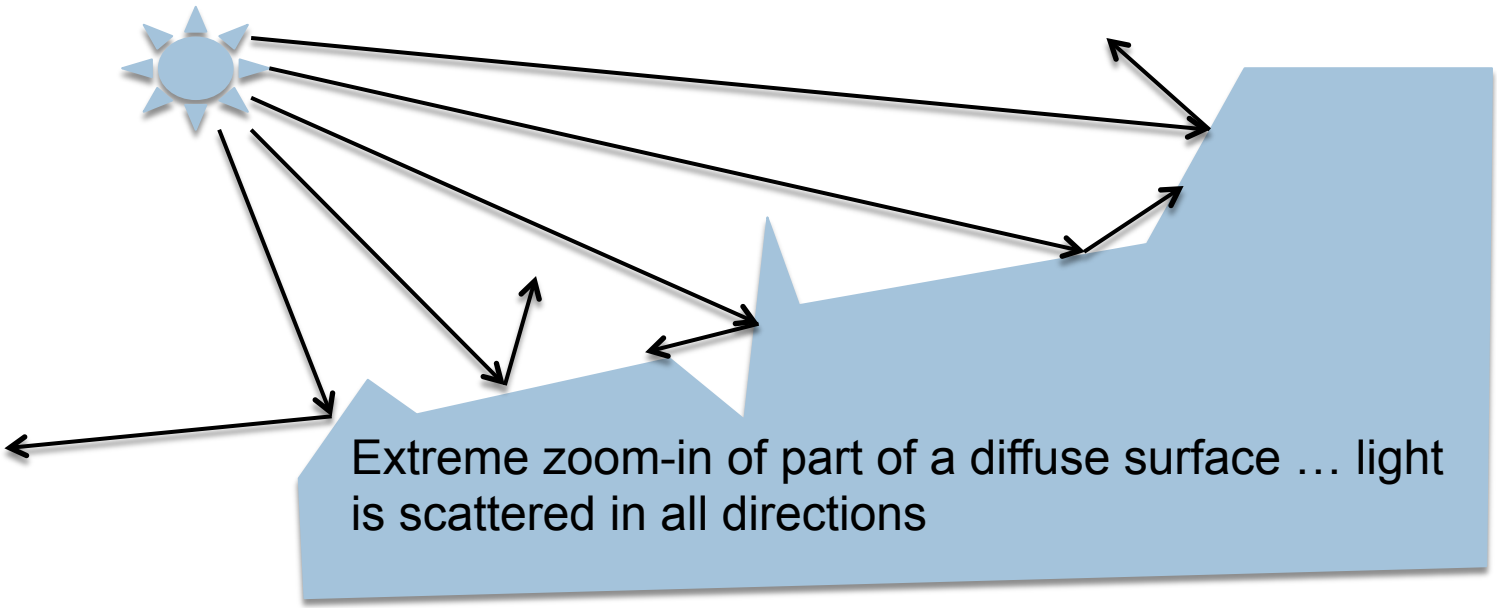
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Surface lit with ambient lighting only



Lambertian Surface

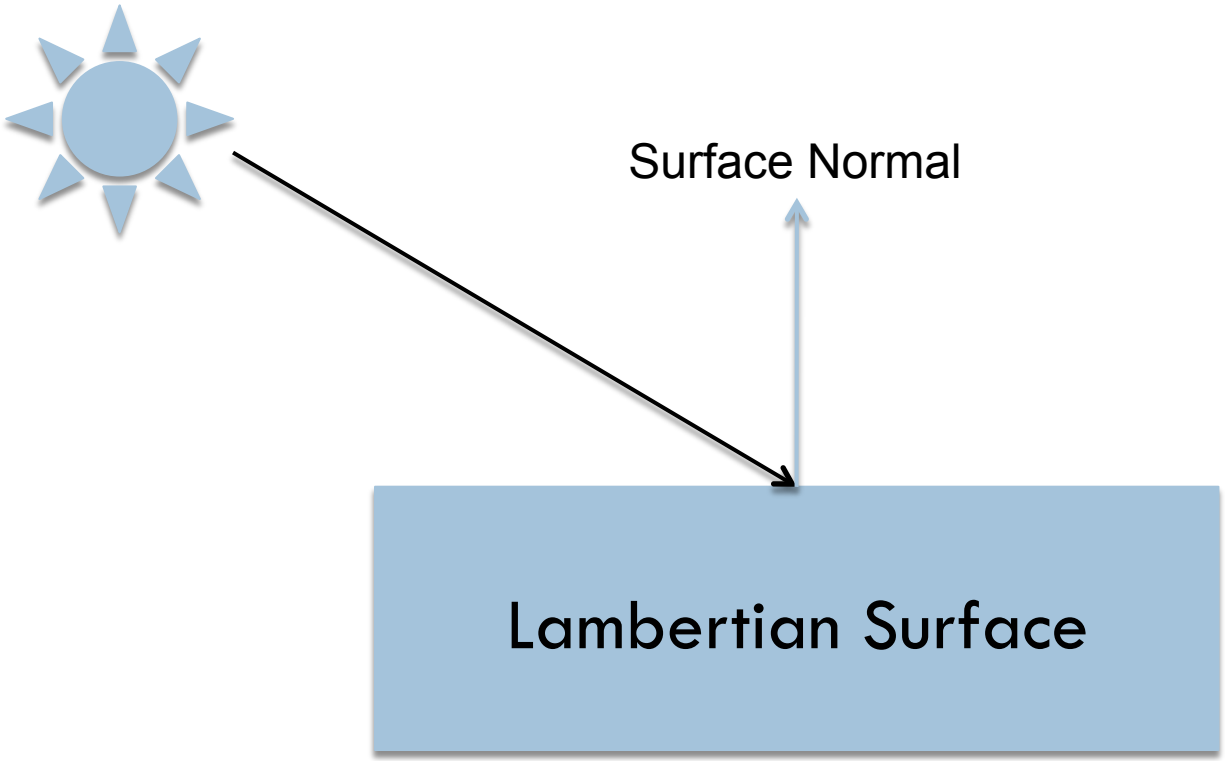
- Perfectly diffuse reflector
- Light scattered equally in all directions



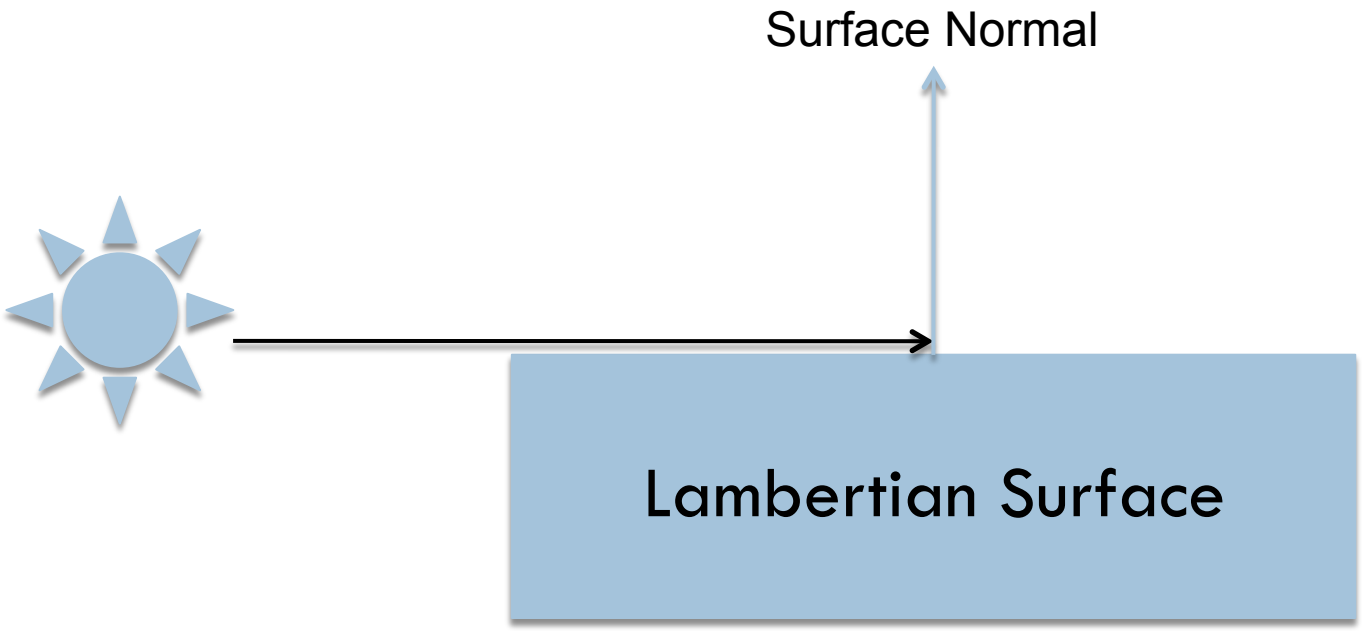
(this image shows 5 of the directions)

Slide inspired by Ed Angel Computer Graphics Book

Diffuse Lighting

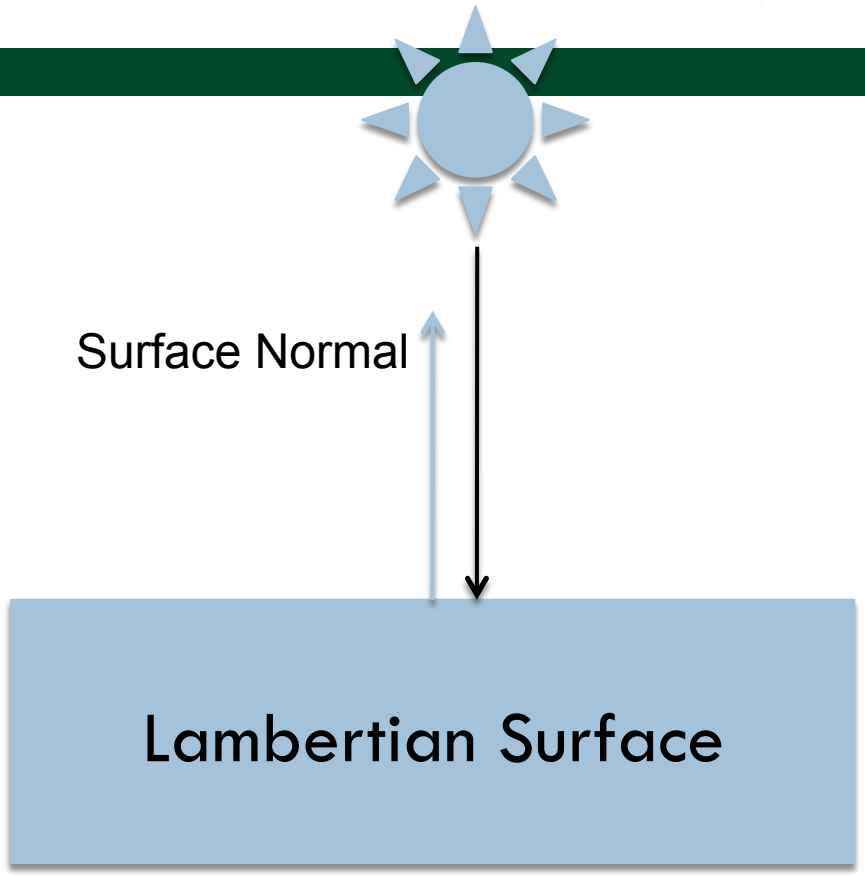


Diffuse Lighting



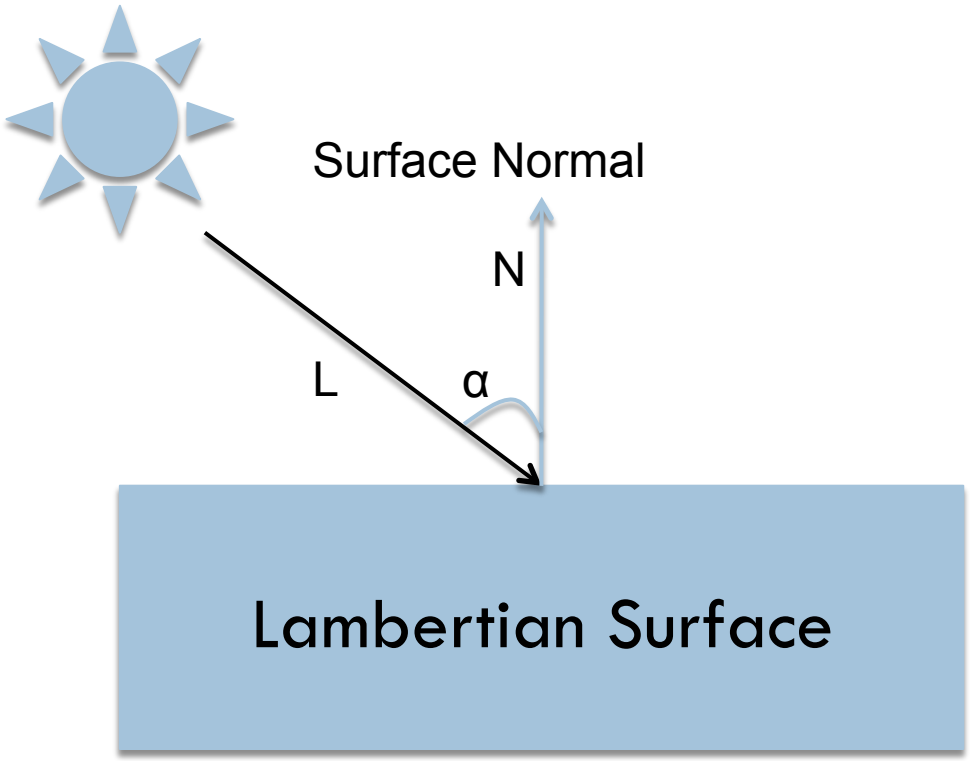
No light reflects off the (top) surface
(Light direction and surface normal are perpendicular)

Diffuse Lighting



When the light squarely hits the surface, then that's when the most light is reflected

Diffuse Lighting



How much light should be reflected in this case?

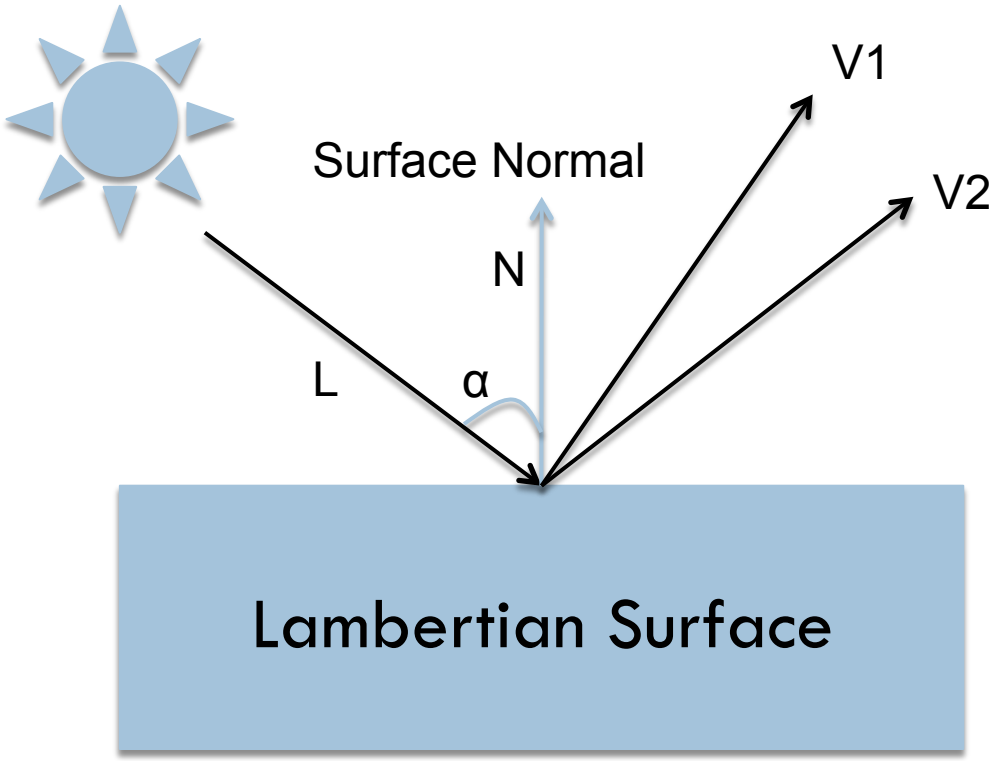
A: cos(α)

And note that:

cos(0) = 1

cos(90) = 0

Diffuse Lighting



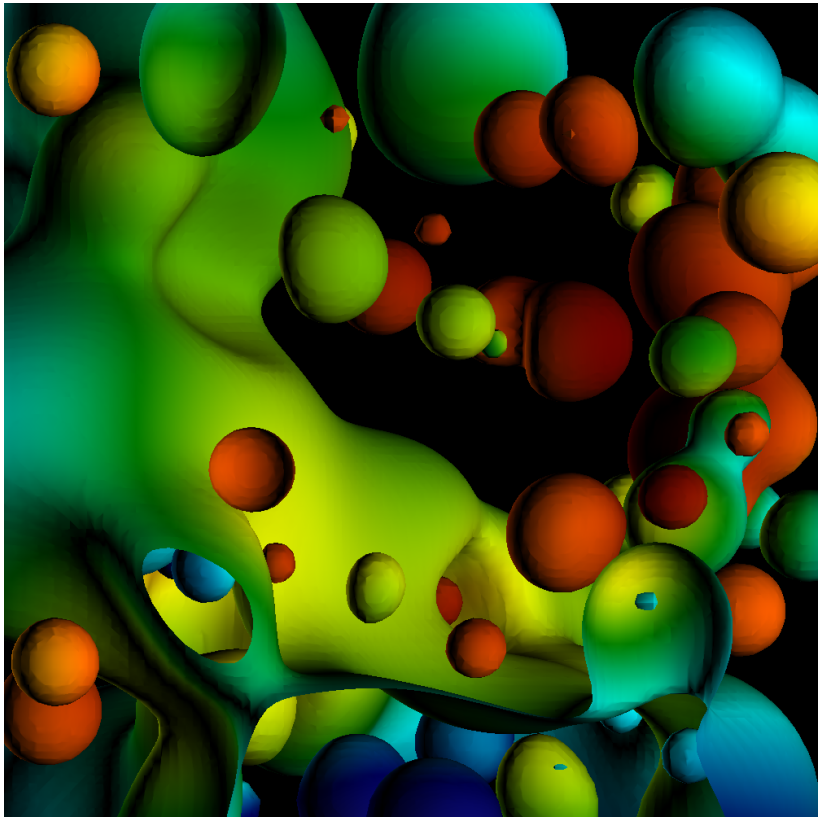
How much light makes it to viewer V1? Viewer V2?

A: cos(α) for both

Lambertian surfaces reflect light equally in all directions

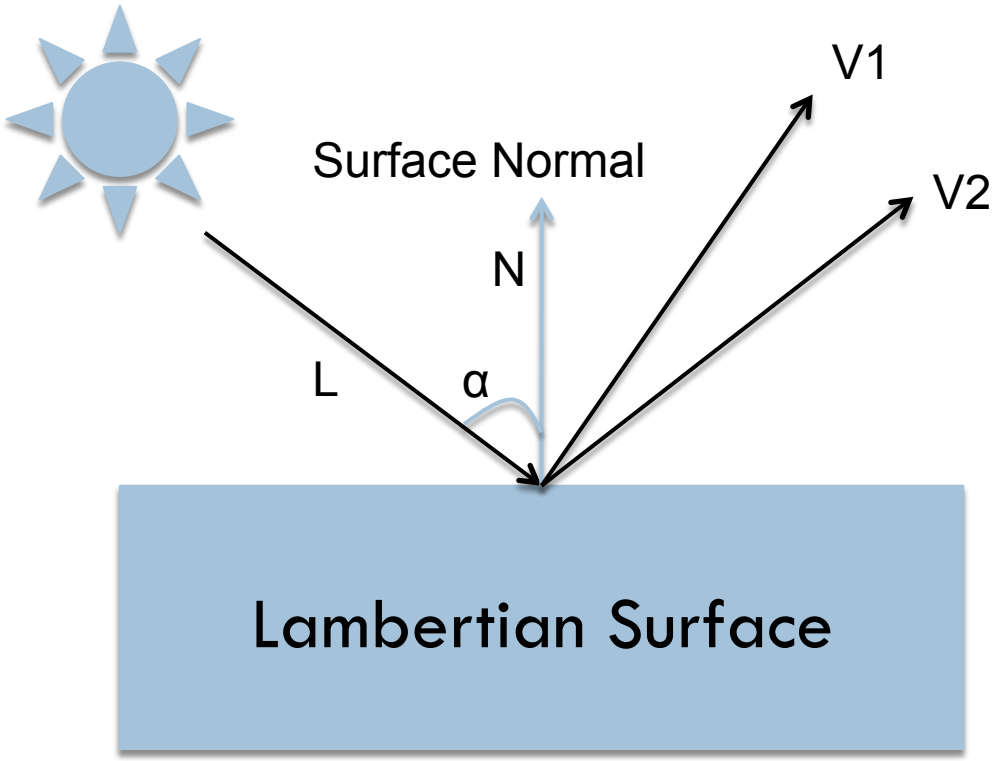
Diffuse Lighting

- Diffuse light
  - Light distributed evenly in all directions, but amount of light depends on orientation of triangles with respect to light source.
  - Different for each triangle



Surface lit with diffuse lighting only

SLIDE REPEAT: Diffuse Lighting



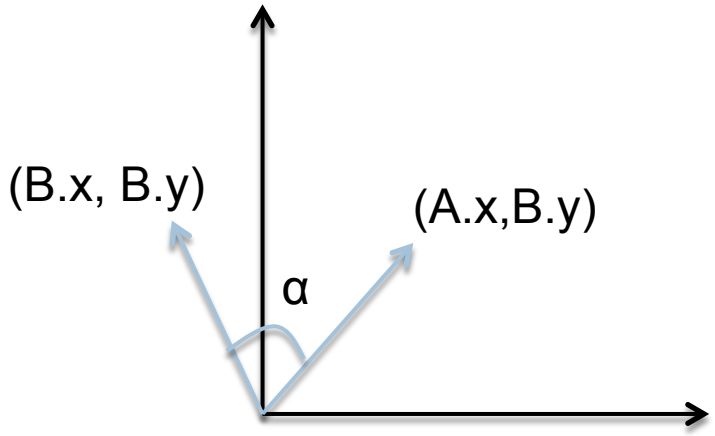
How much light makes it to viewer V1? Viewer V2?

A: cos(α) for both

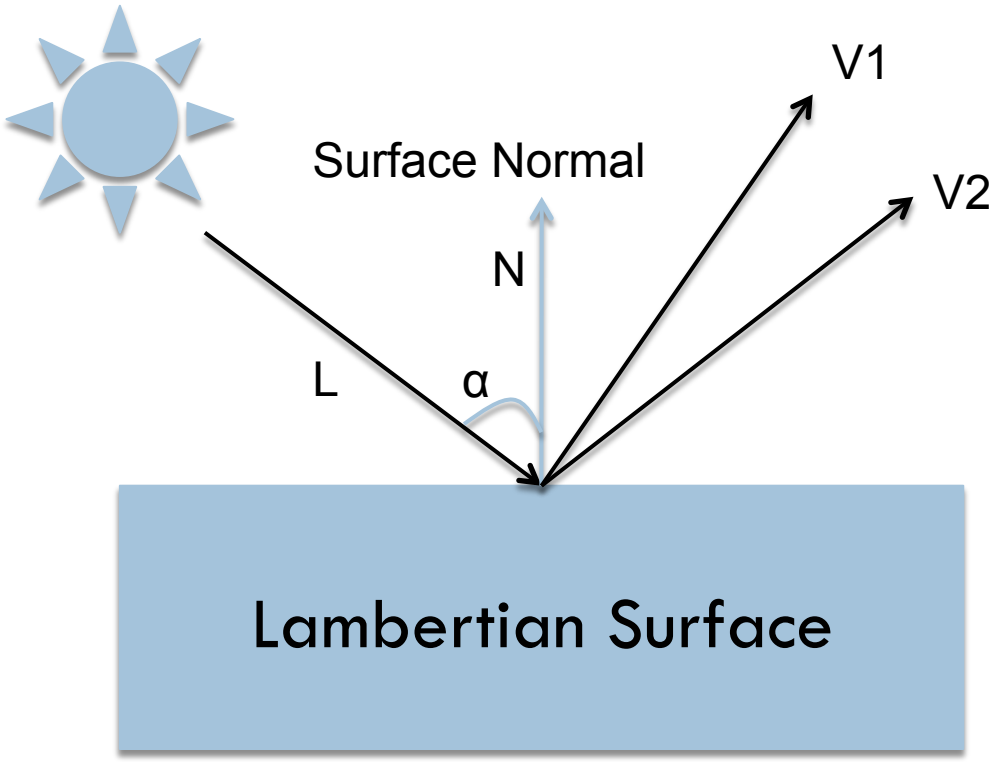
Lambertian surfaces reflect light equally in all directions

What is a dot product?

- $A \cdot B = A.x*B.x + A.y*B.y$
- Physical interpretation:
  - $A \cdot B = \cos(\alpha)/(||A||*||B||)$

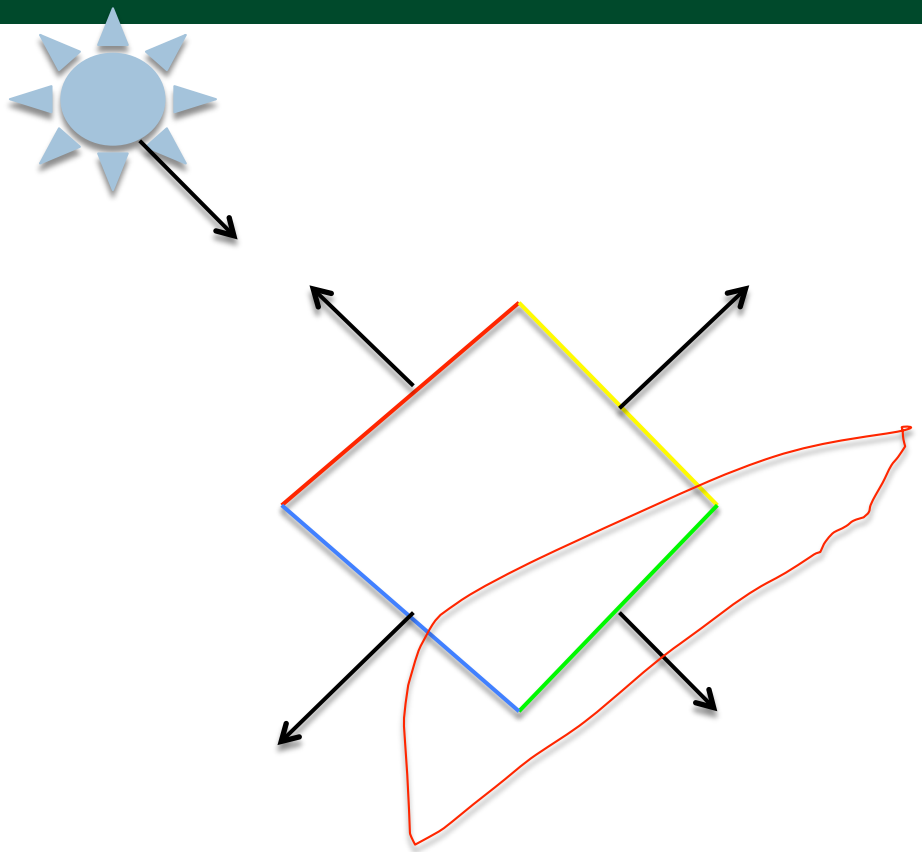


Diffuse Lighting

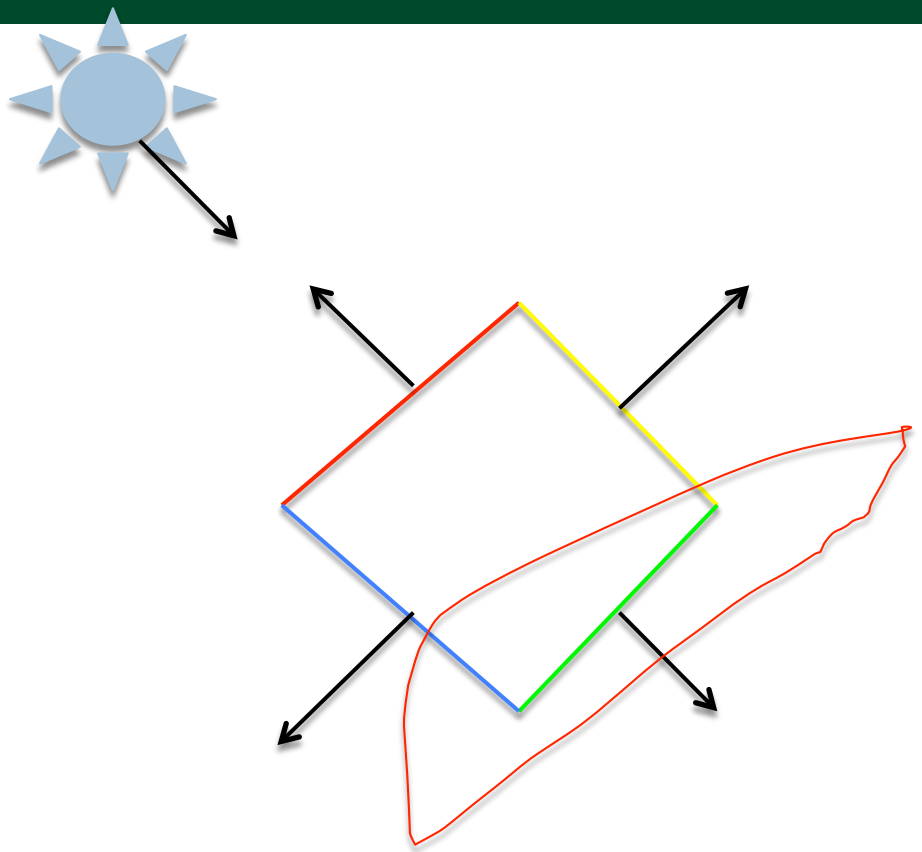


You can calculate the diffuse contribution by taking the dot product of L and N,
Since $L \cdot N = \cos(\alpha)$
(assuming L and N are normalized)

What about cases where L·N < 0?

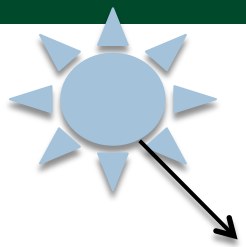


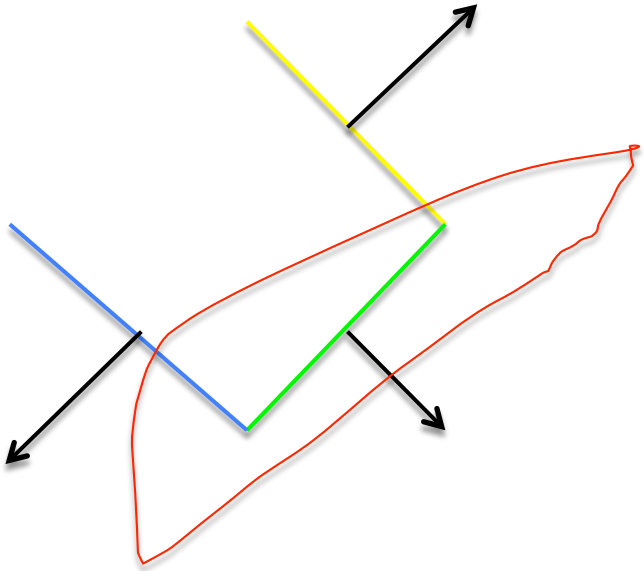
What about cases where L·N < 0?



L·N = -1
Non-sensical … takes away light?
Common solution:
Diffuse light = max(0, L·N)

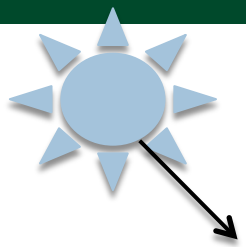
But wait…

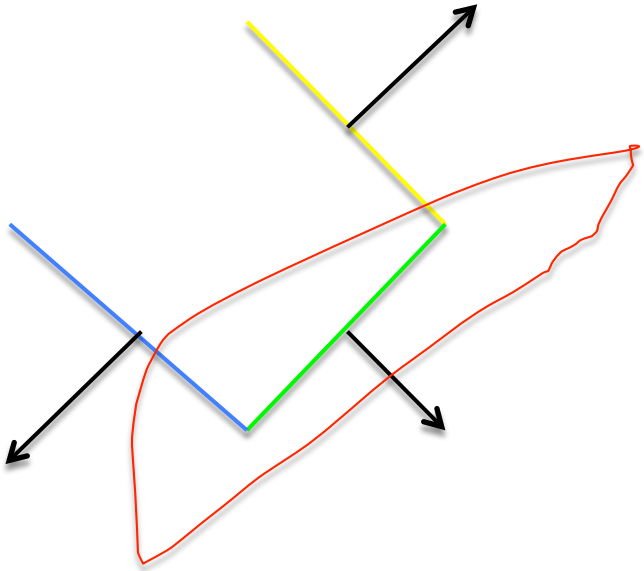




If you have an open surface, then there is a “back face”. The back face has the opposite normal.

But wait…

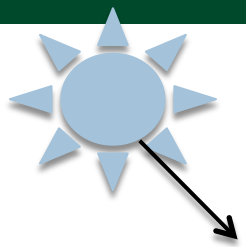


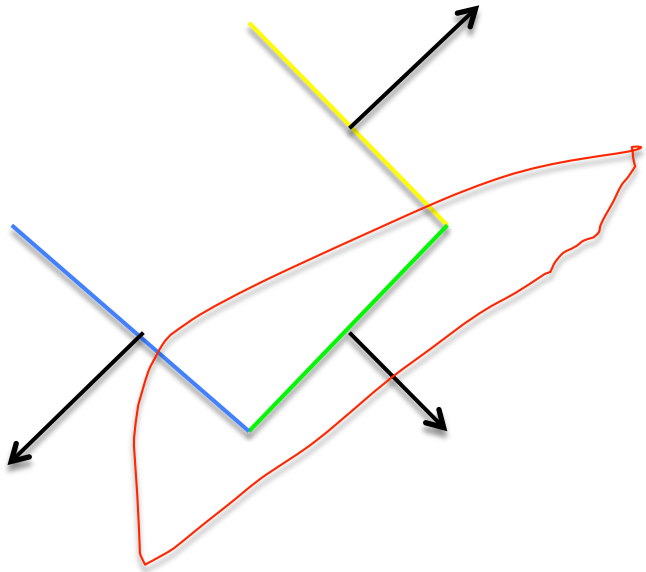


If you have an open surface, then there is a “back face”. The back face has the opposite normal.

How can we deal with this case?

But wait…



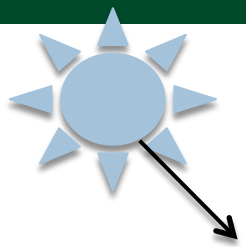


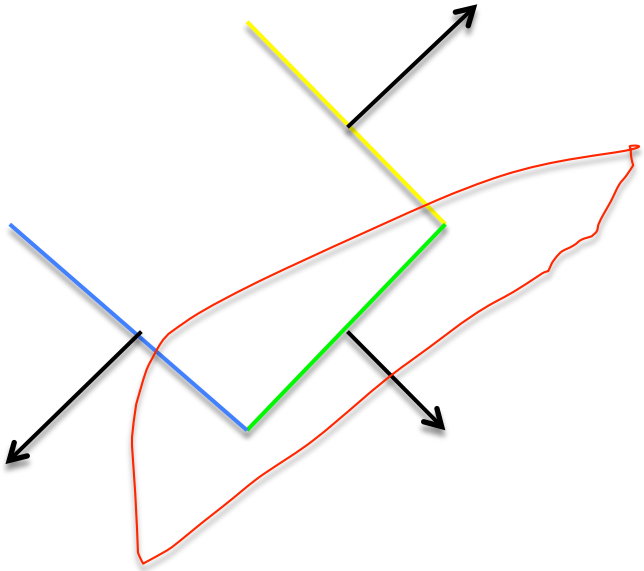
If you have an open surface, then there is a “back face”. The back face has the opposite normal.

How can we deal with this case?

Idea #1: encode all triangles twice, with different normals
Idea #2: modify diffuse lighting model

But wait…





If you have an open surface, then there is a “back face”. The back face has the opposite normal.

How can we deal with this case?

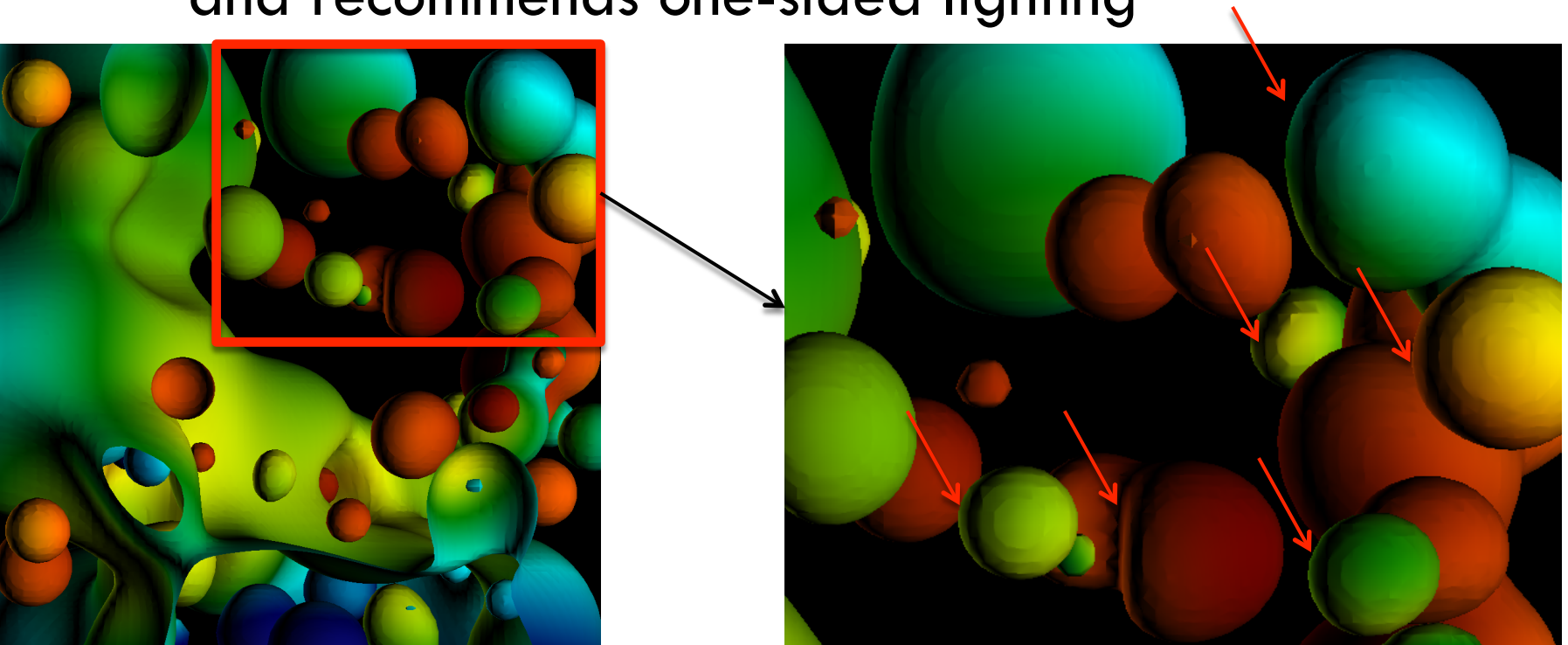
Idea #1: encode all triangles twice, with different normals
Idea #2: modify diffuse lighting model

Diffuse light = abs(L·N)

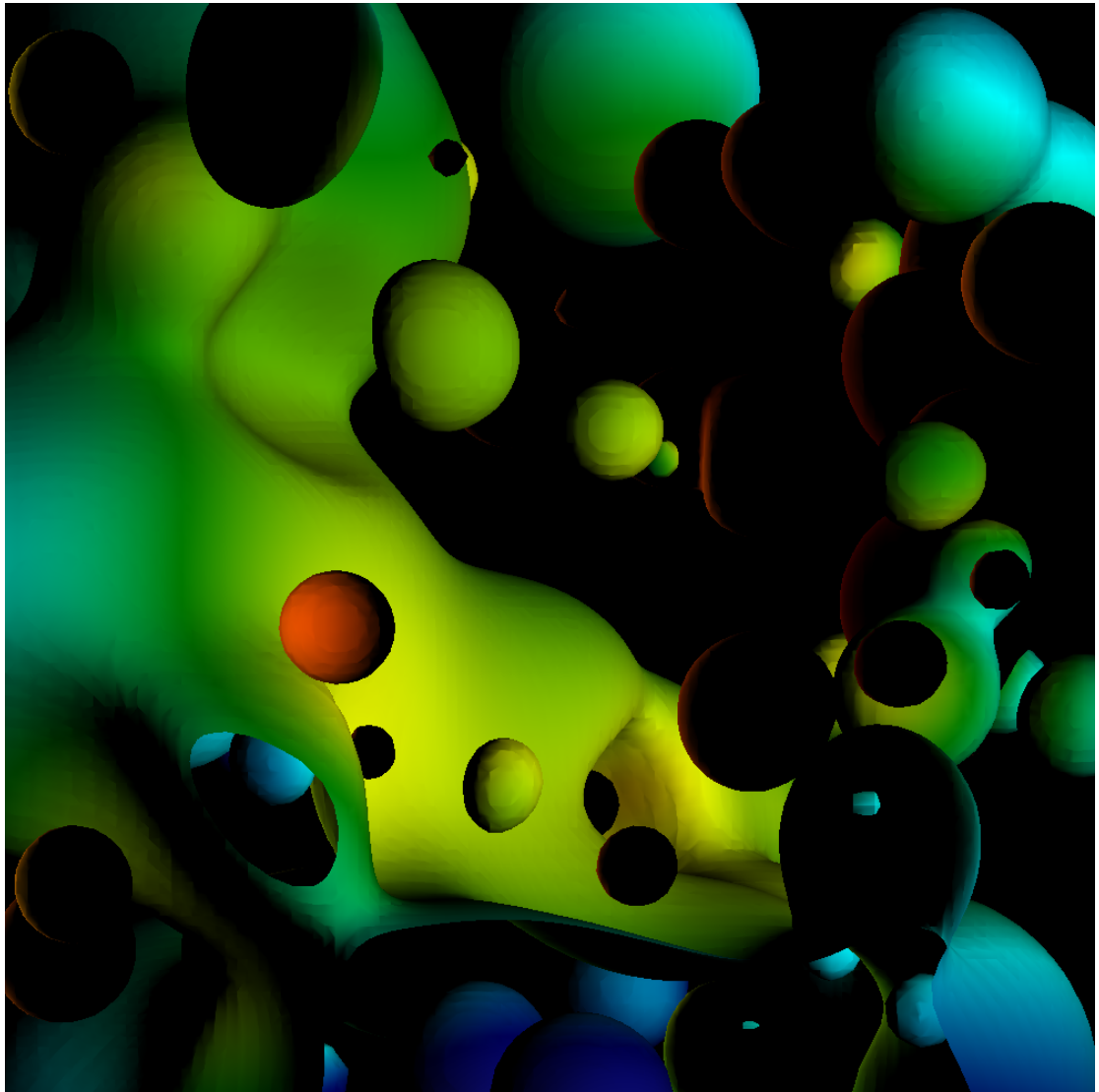
This is called two-sided lighting

Two-sided lighting

- We will use two-sided lighting for project 1E, since we have open surfaces
- Note that Ed Angel book assumes closed surfaces and recommends one-sided lighting

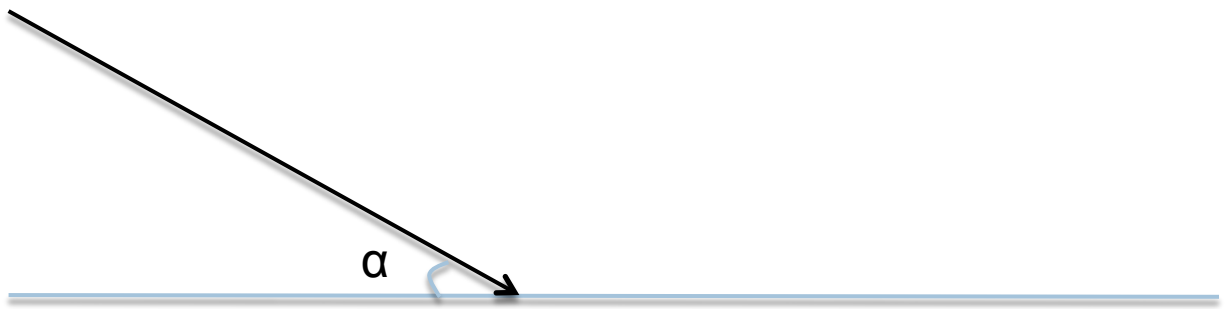


One-sided lighting with open surfaces is disappointing



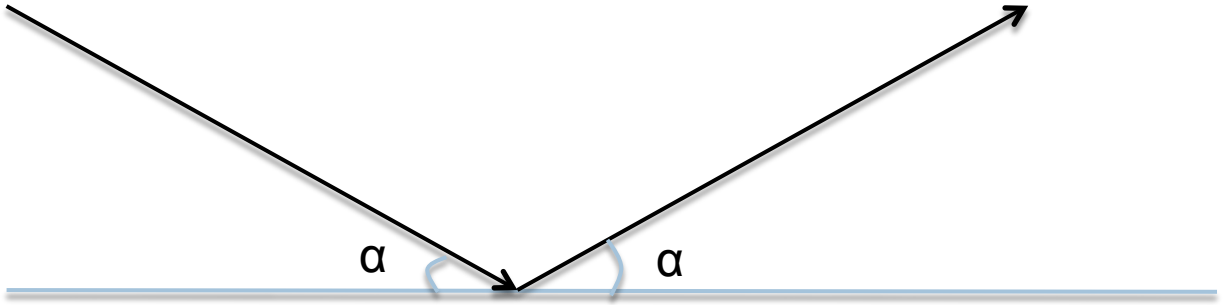
The most valuable thing I learned in Freshman Physics

- "angle in = angle out"

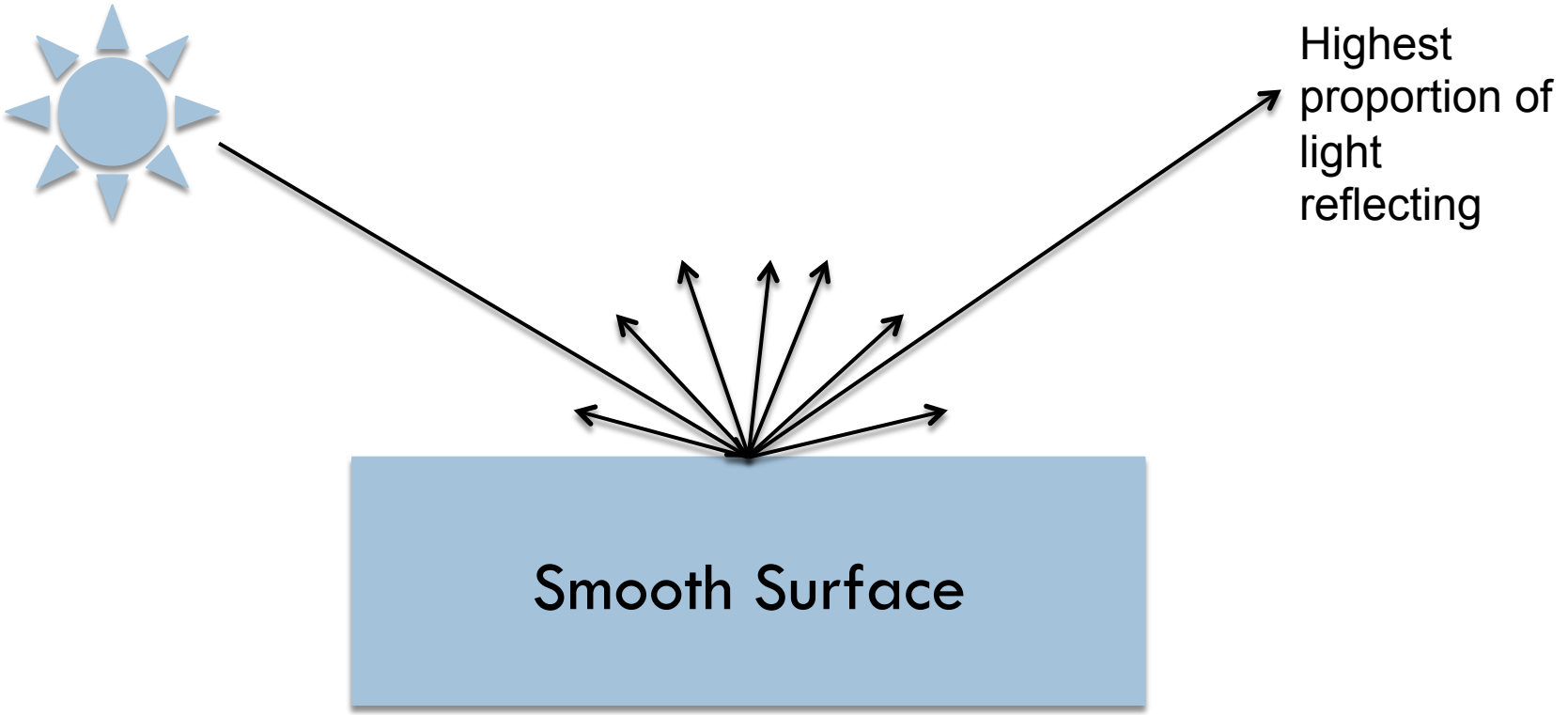


The most valuable thing I learned in Freshman Physics

- "angle in = angle out"

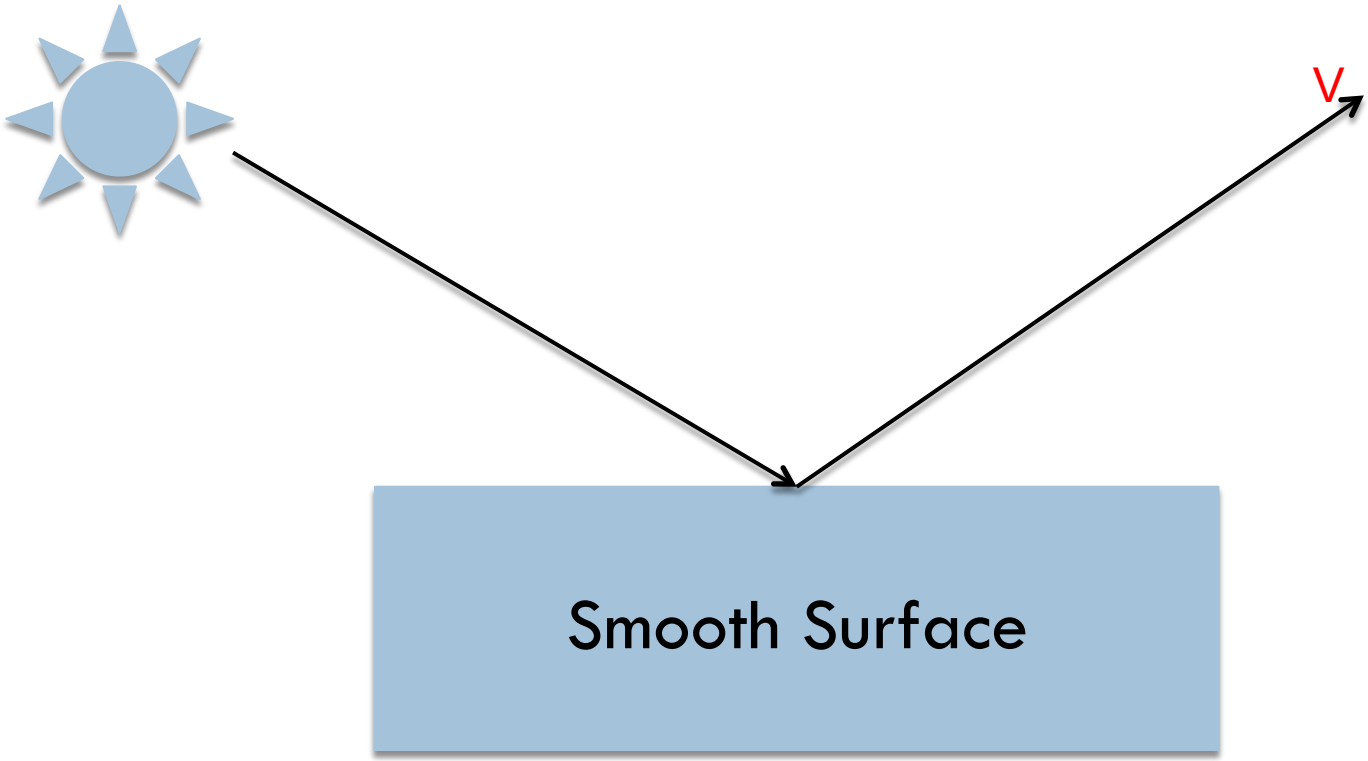


Specular Lighting



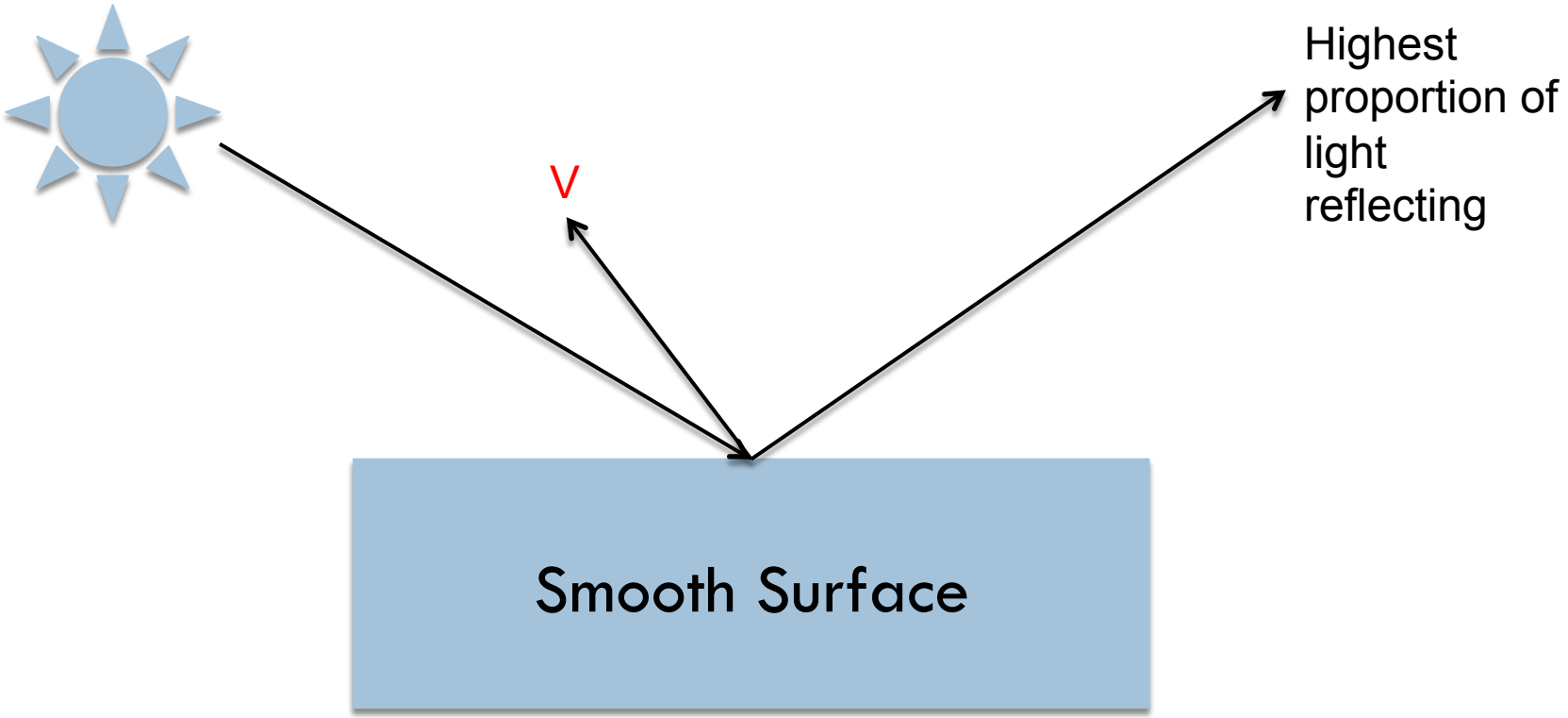
Light reflects in all directions.
But the surface is smooth, not Lambertian, so amount of reflected light varies.
So how much light??

How much light reflects with specular lighting?



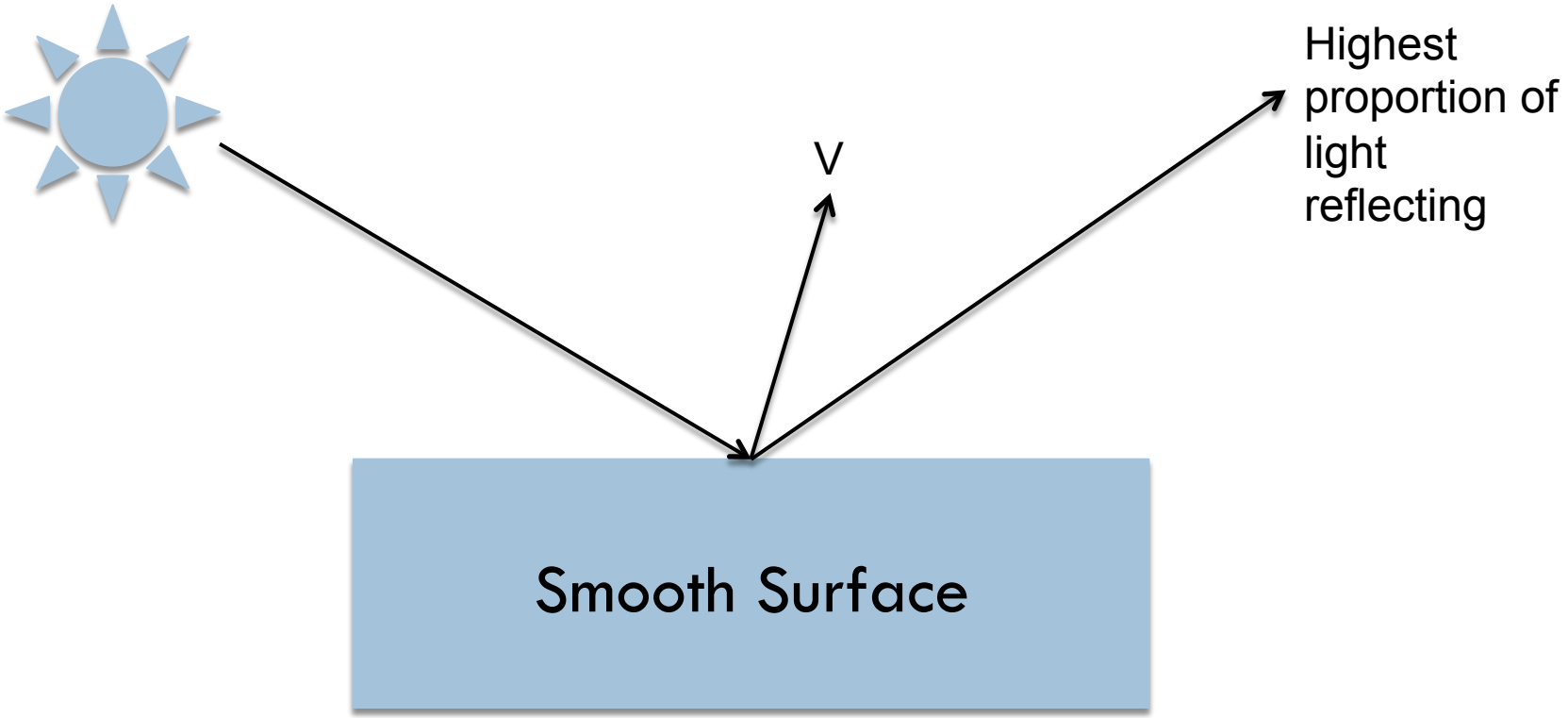
Consider V located along reflection ray.
Answer: most possible
Call this “1”

How much light reflects with specular lighting?



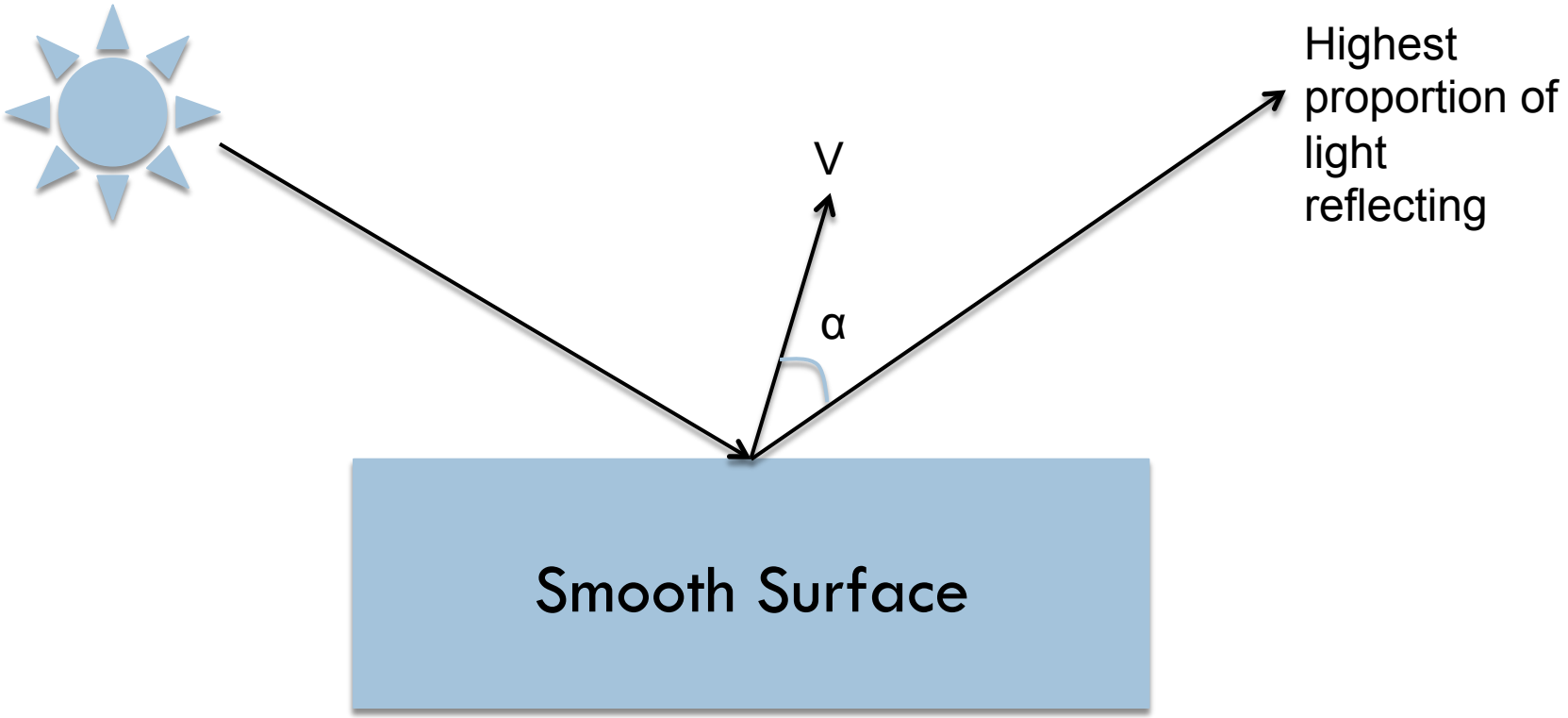
Consider V located along perpendicular ray.
Answer: none of it
Call this “0”

How much light reflects with specular lighting?



How much light gets to point V?

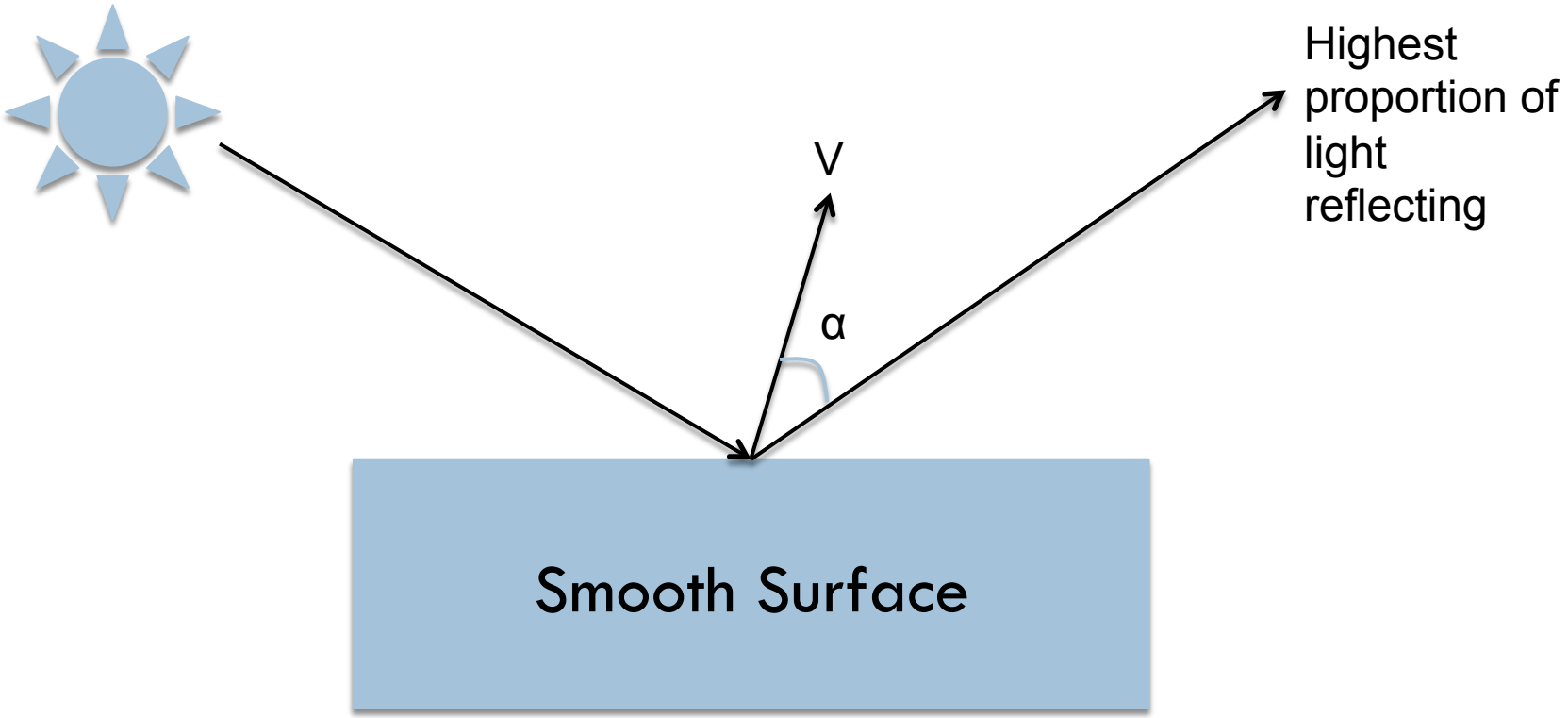
How much light reflects with specular lighting?



How much light gets to point V?

A: proportional to cos(α)

How much light reflects with specular lighting?

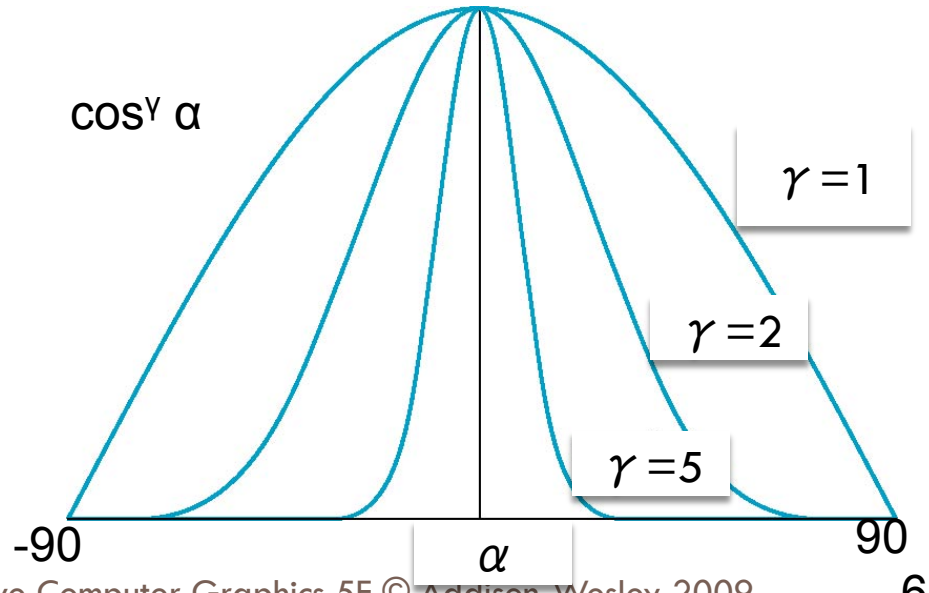


How much light gets to point V?

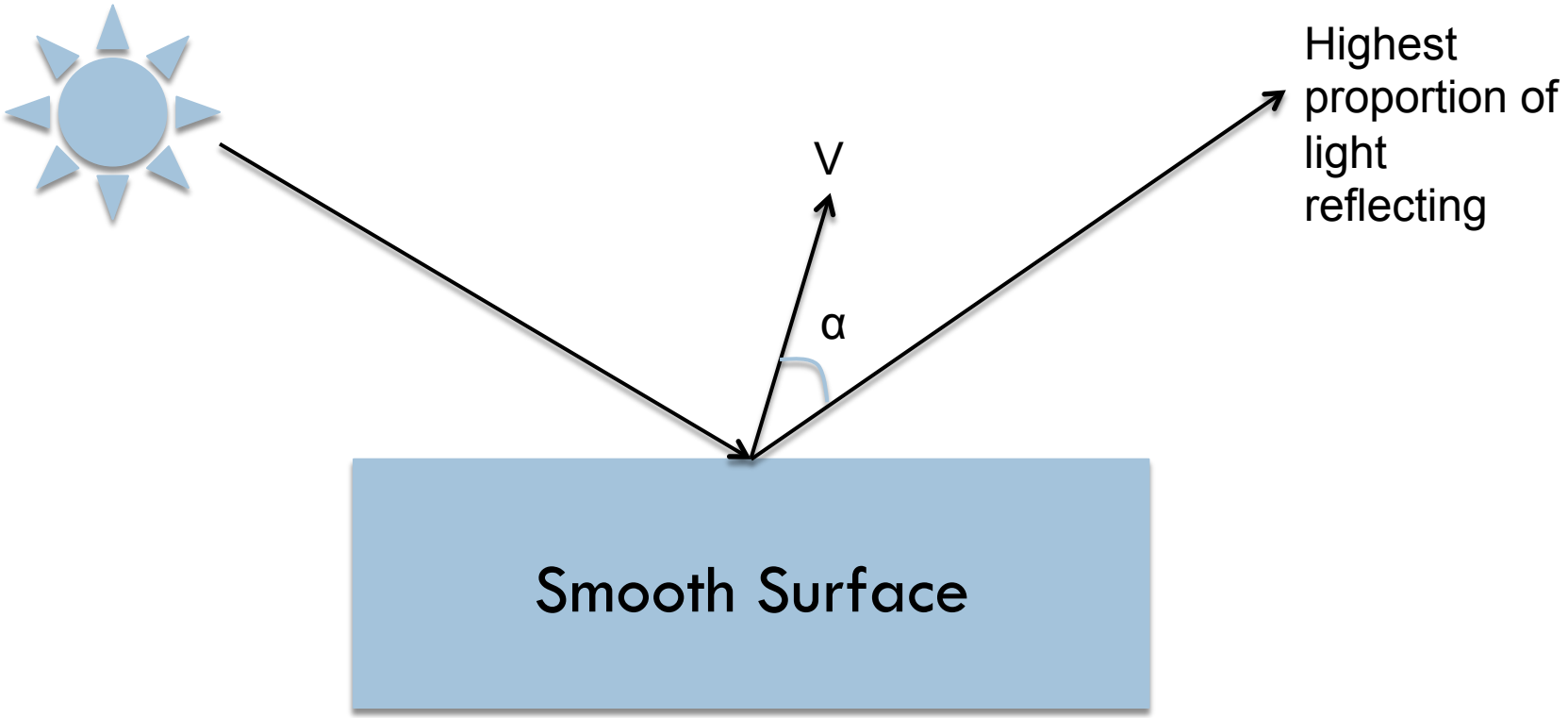
A: proportional to cos(α)
(Shininess strength) * cos(α) ^ (shininess coefficient)

γ : The Shininess Coefficient

- Values of γ between 100 and 200 correspond to metals
- Values between 5 and 10 give surface that look like plastic



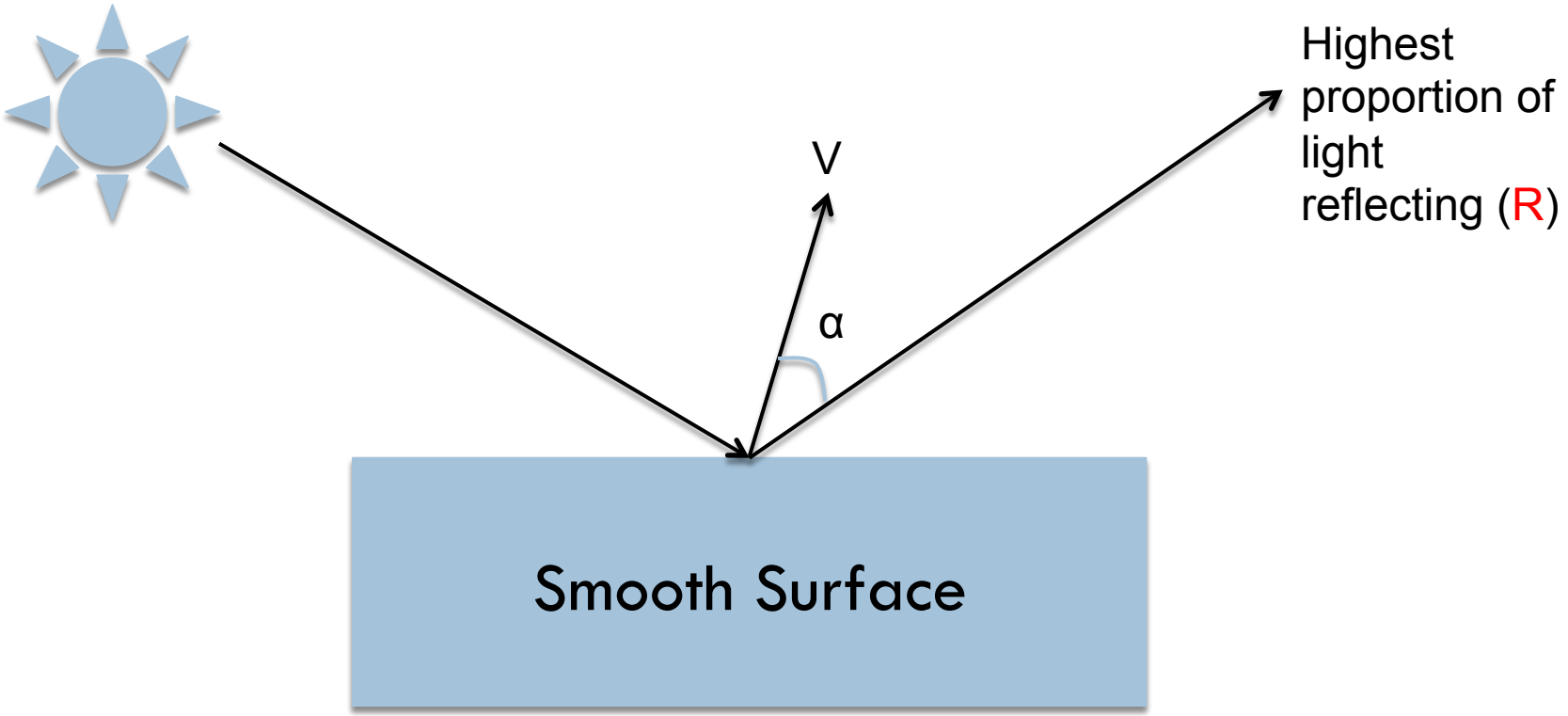
How much light reflects with specular lighting?



How much light gets to point V?

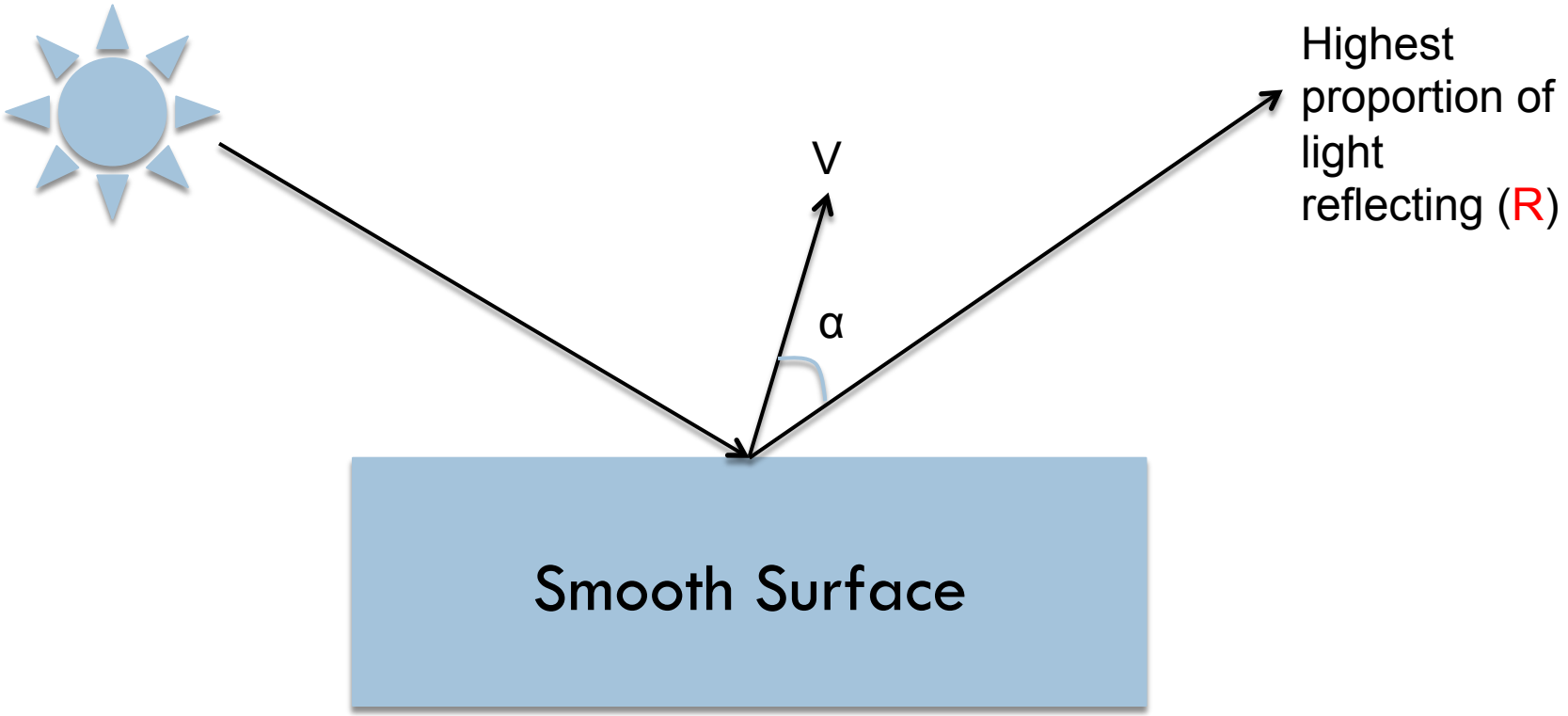
A: proportional to cos(α)
(Shininess strength) * cos(α) ^ (shininess coefficient)

How much light reflects with specular lighting?



Great!
We know that cos(α) is V·R.

How much light reflects with specular lighting?



Great!
We know that cos(α) is V·R.
But what is R?
It is a formula: R = 2*(L·N)*N - L

Two-sided lighting

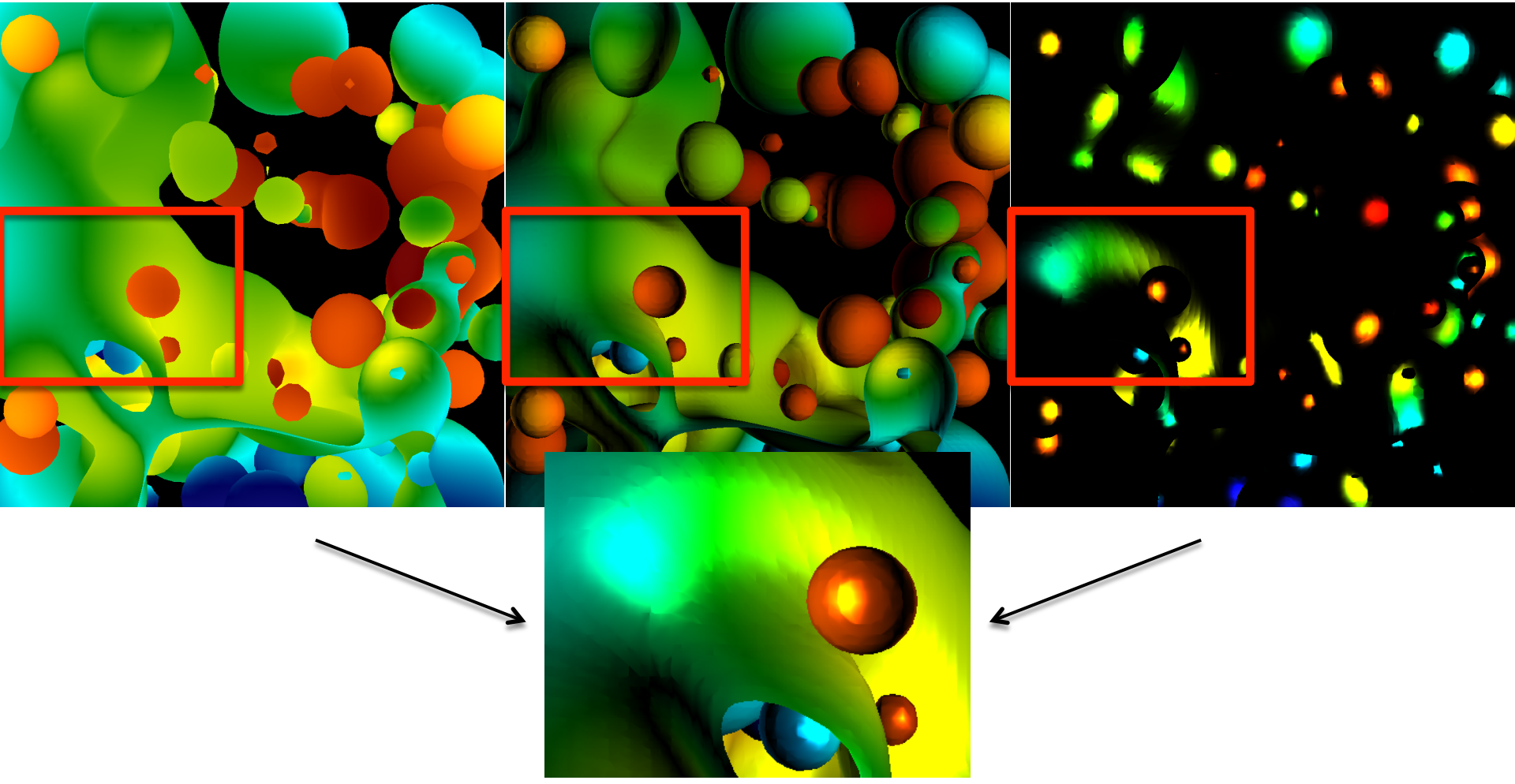
- For specular lighting, we will use one-sided lighting for project 1E
  - It just looks better
  - Diffuse: abs(L·N)
  - Specular: max(0, S*(R·V)$^{\gamma}$)

Outline

- Math Basics
- Lighting Basics
- The Phong Model

Phong Model

- Combine three lighting effects: ambient, diffuse, specular



Phong Model

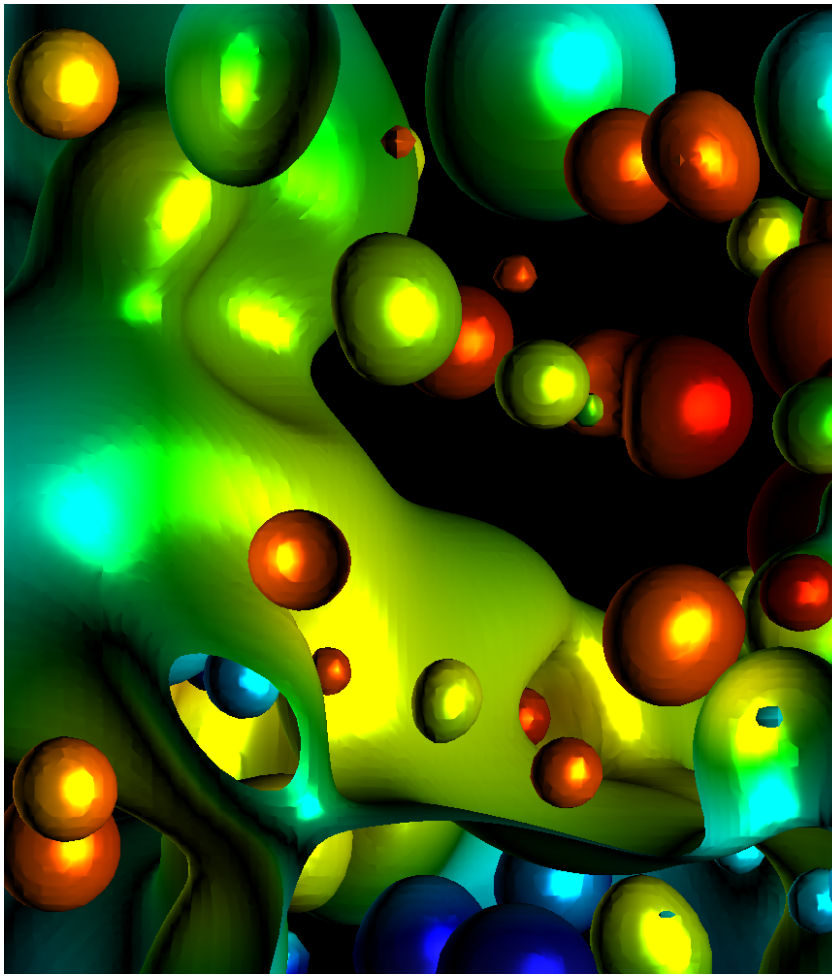
- Simple version: 1 light, with "full intensity" (i.e., don't add an intensity term)
- Phong model
  - Shading_Amount = K_a + K_d*Diffuse + K_s*Specular
- Signature:
  - double CalculatePhongShading(LightingParameters &, double *viewDirection, double *normal)
  - For us, viewDirection = (0,0, +1)
    - or is it?...

Lighting parameters

```
struct LightingParameters
{
    LightingParameters(void)
    {
        lightDir[0] = -0.6;
        lightDir[1] = 0;
        lightDir[2] = -0.8;
        Ka = 0.3;
        Kd = 0.7;
        Ks = 5.3;
        alpha = 7.5
    };


    doiuble lightDir[3]; // The direction of the light source
    double Ka;           // The coefficient for ambient lighting.
    double Kd;           // The coefficient for diffuse lighting.
    double Ks;           // The coefficient for specular lighting.
    double alpha;        // The exponent term for specular lighting.
};
```

Project #1E (7%) (soon)

- Goal: add Phong shading
- Extend your project1D code
- File proj1e_geometry.vtk available on web (9MB)
- File “reader1e.cxx” has code to read triangles from file.
- No Cmake, project1e.cxx



Changes to data structures

```
class Triangle
{
  public:
      double X[3], Y[3], Z[3];
      double colors[3][3];
      double normals[3][3];
};
```

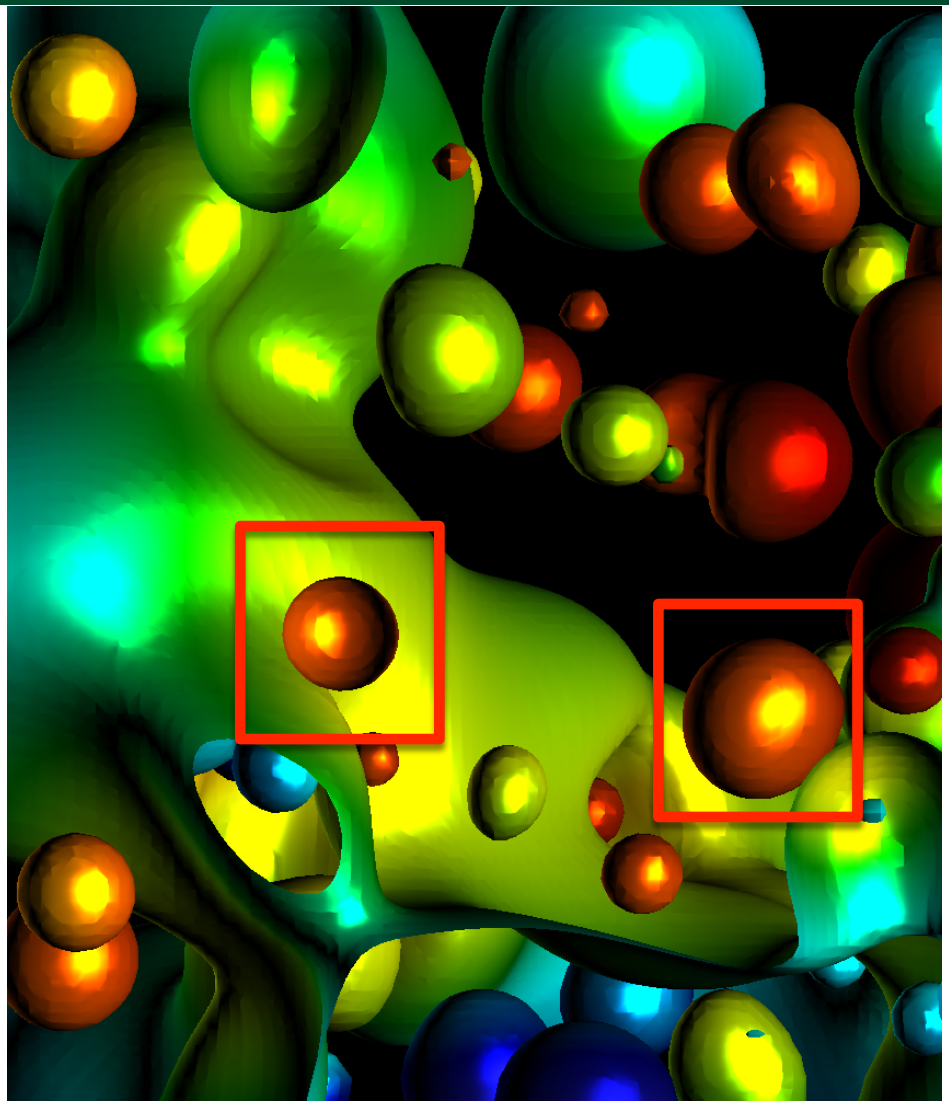
→reader1e.cxx will not compile until you make these changes

→reader1e.cxx will initialize normals at each vertex

More comments

- New: more data to help debug
  - I will make the shading value for each pixel available.
  - I will also make it available for ambient, diffuse, specular.
- Don't forget to do two-sided lighting
- This project in a nutshell:
  - LERP normal to a pixel
    - You all are great at this now!!
  - Add method called "CalculateShading".
    - My version of CalculateShading is about ten lines of code.
  - Modify RGB calculation to use shading.

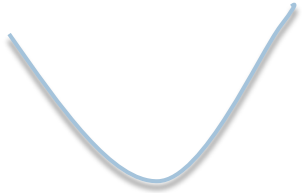
Where Hank spent his debugging time…



Concave surface

Lighting direction

Convex surface



Lighting direction