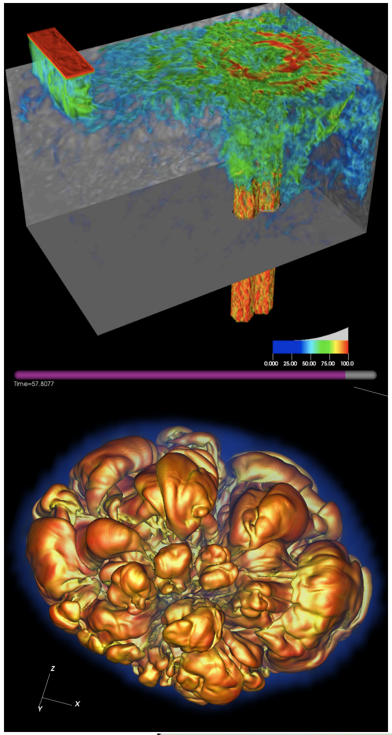
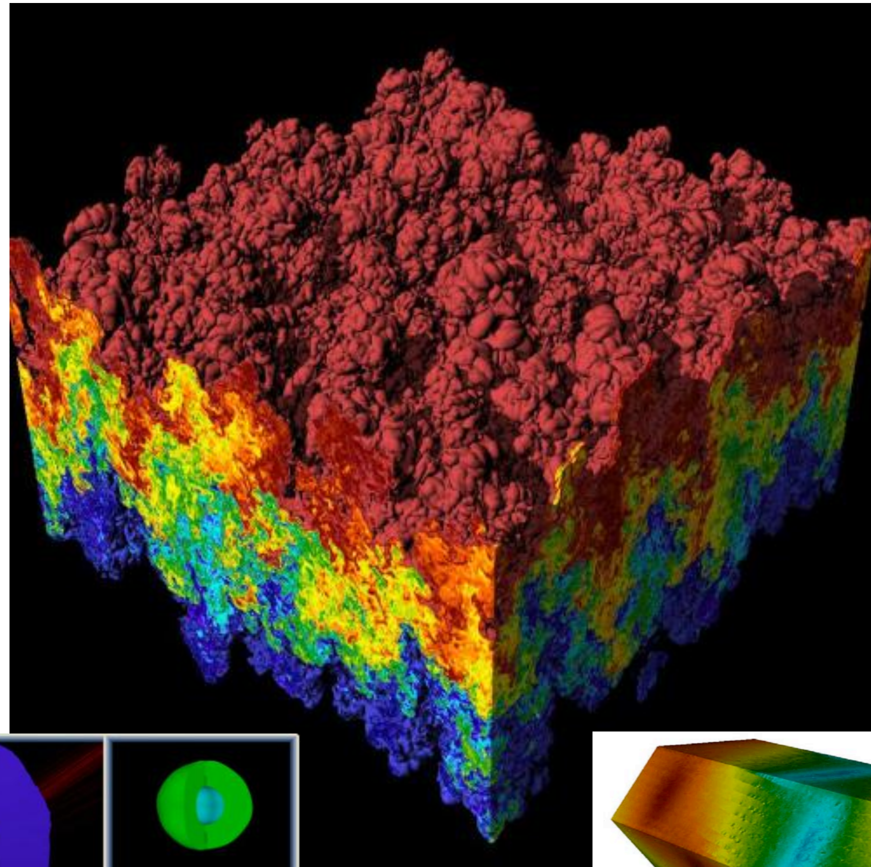
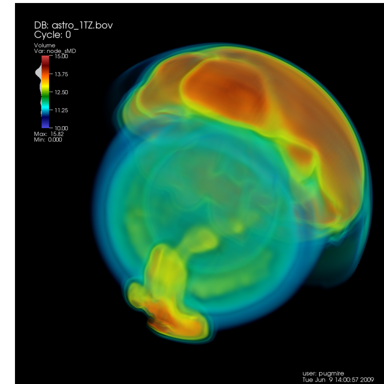
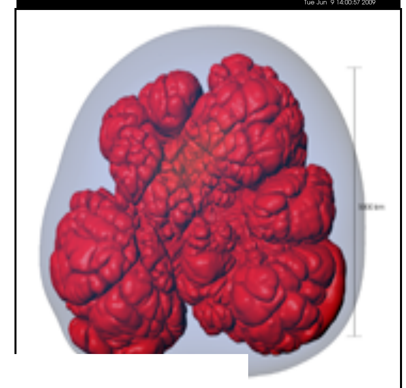
UNIVERSITY OF OREGON

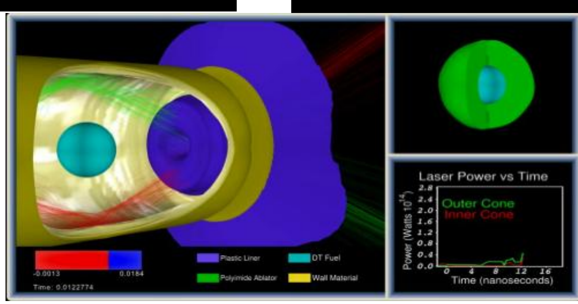
CIS 441/541: Intro to Computer Graphics
Lecture 7: Math Basics, Lighting Introduction & Phong Lighting

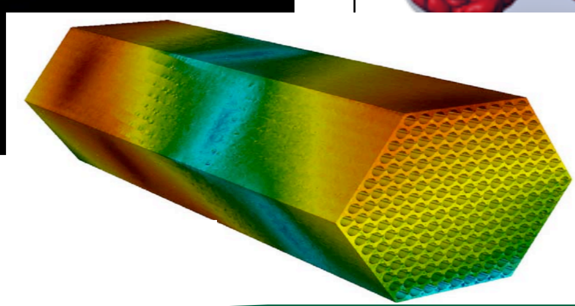












January 31, 2019

Hank Childs, University of Oregon

Office Hours: Weeks 4-10

- Monday: 1-2 (Roscoe)
- Tuesday: 1-2 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: 1130-1230 (Hank)
- Friday: 1130-1230 (Hank)

- All normal this week!!! ☺

Timeline

- ~~1C: due Weds Jan 23rd~~
- 1D: ~~assigned today (LAST TUESDAY),~~ due Thurs Jan 31st
- 1E: assigned Thurs Jan 31st, due Weds Feb 6th
  - → will be extra support with this. Tough project.
- 1F: assigned Feb 7th (probably before), due Feb 19th
  - → not as tough as 1E
- 2A: will be assigned during week of Feb 11th (maybe before)

Sun	Mon	Tues	Weds	Thurs	Fri	Sat
Jan 20	Jan21	Jan 22 Lec4	Jan 23 1C due	Lec 5 1D assigned	Jan 25	Jan 26
Jan 27	Jan 28	Jan 29 (YouTube)	Jan 30	Lec 6 1D due 1E assigned	Feb 1	Feb 2
Feb 3	Feb 4	Feb 5 Lec 7	Feb 6 1E due	Feb 7 1F assigned	Feb 8	Feb 9

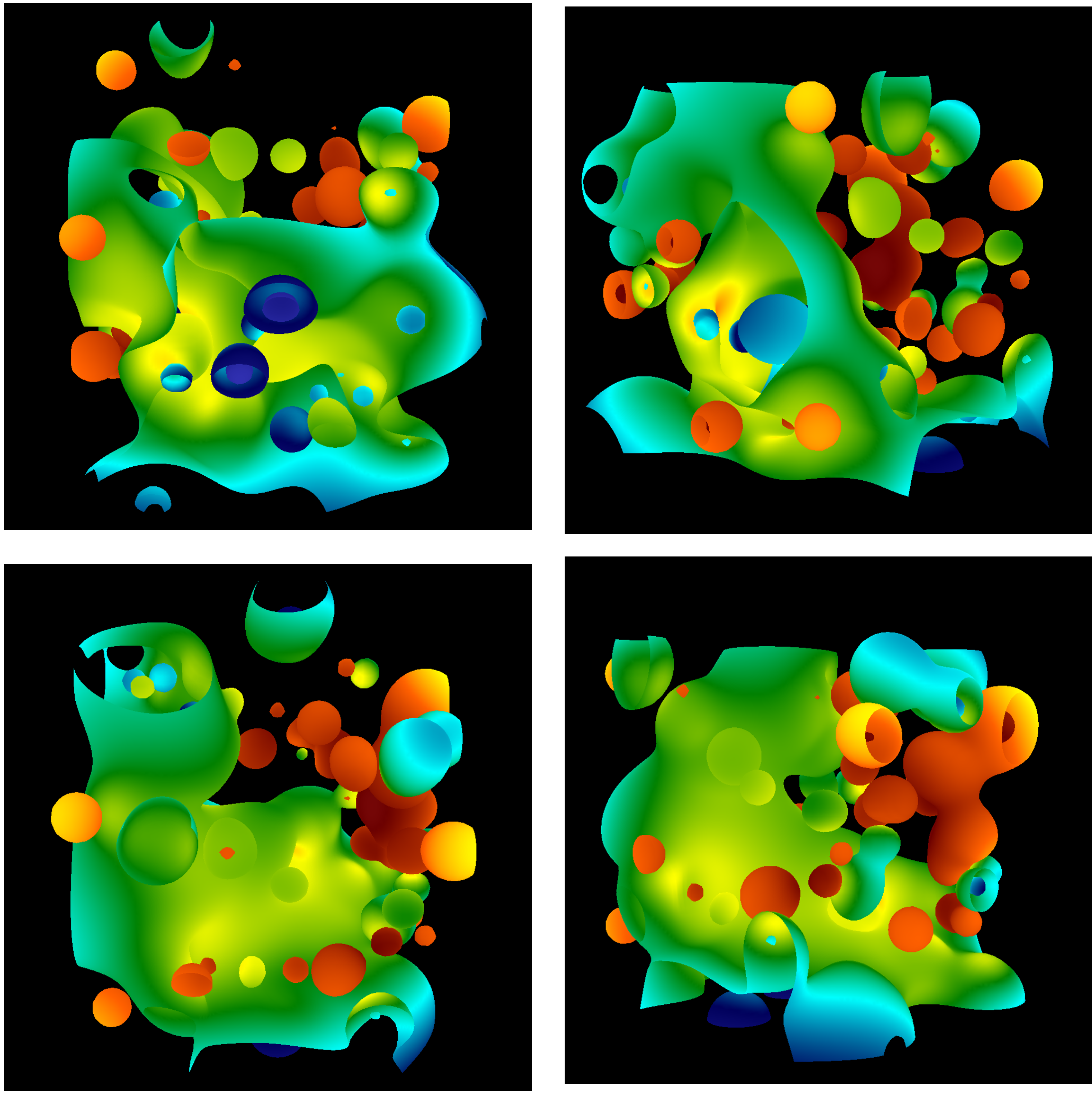
Likely: pre-SuperBowl OH

Sunday OH?

- Sunday Feb 3rd: 1030-1145
- ???

Project #1E (6%), Due Weds Feb 6th

- Goal: add arbitrary camera positions
- Extend your project1D code
- New: proj1e_geometry.vtk available on web (9MB), "reader1e.cxx".
- New: Matrix.cxx, Camera.cxx
- No Cmake, project1E.cxx
- QUESTIONS ON 1E?



New topic: Hank's travel

From Lecture #1: Planned Absences

Workshop on In Situ Data Management

Sponsored by the U.S. Department of Energy, Office of Advanced Scientific Computing Research (ASCR)
North Bethesda, MD
January 28 – 29, 2019

NII SHONAN MEETING

News » Committee Call for Proposals » Coming Seminars » Reports FAQ »

Call for Prop

We are constantly accept
Next due date is Decemb
seminars we are calling for, are those to be held between 8 months ahead and 2 years ahead of the next submission due date. (Example: In the case that the next due date is December 15th, 2017, we are calling for the seminar to be held from August...

Read More

How should we deal with Hank's travel?

- We are halfway through my travel commitments
- What should we do for the remainder?
  - Guest lectures
  - Video lectures + OH
  - One of each?
- (Are video lectures working?)

Outline

- Math Basics
- Lighting Basics
- The Phong Model

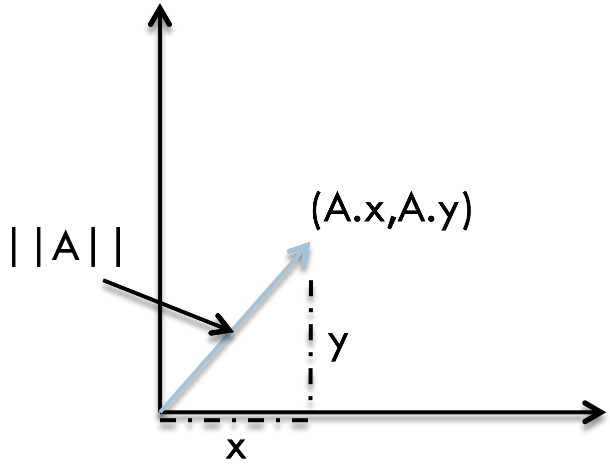
Outline

- Math Basics
- Lighting Basics
- The Phong Model

What is the norm of a vector?

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

  ||A|| = sqrt(A.x*A.x+A.y*A.y)
- Physical interpretation:



- For 3D, ||A|| = 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.
- Example: A=(3,4,0). Then:
  - $||A|| = 5$
  - $A/||A|| = (0.6, 0.8, 0)$

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,

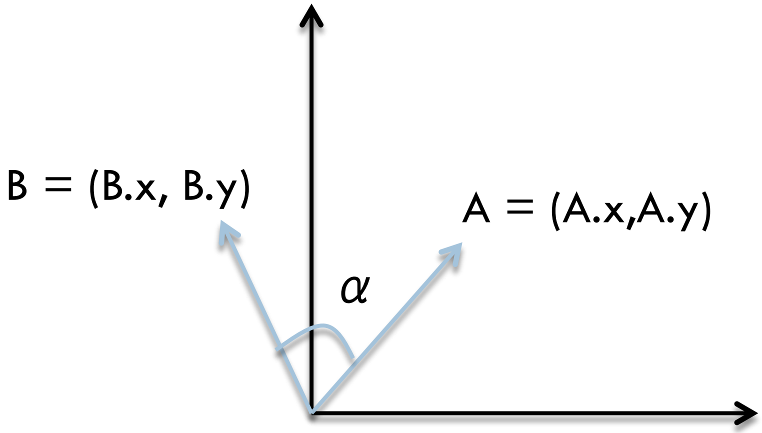
  then the triangle's normal is (A, B, C)

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·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·B = cos(α)*(||A||*||B||)



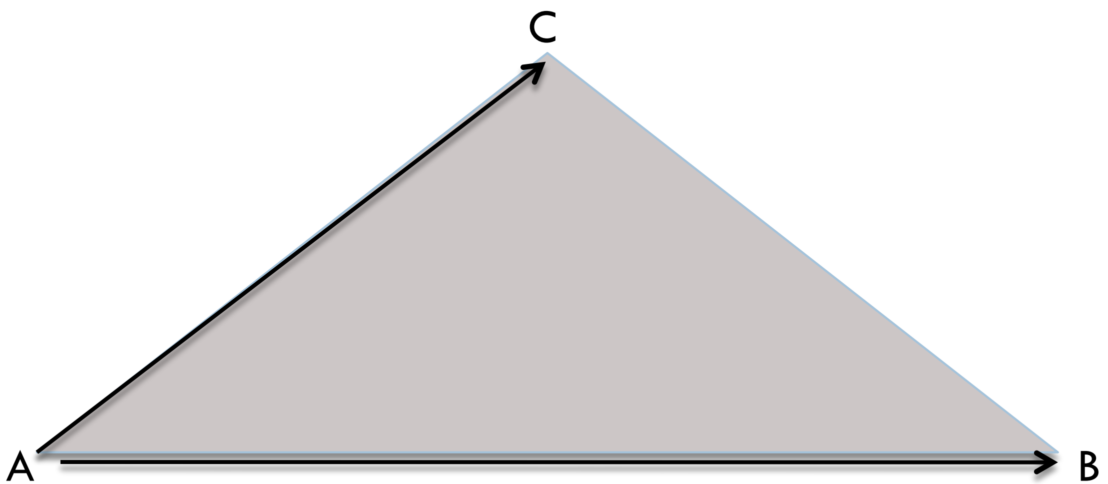
What is the cross product?

- 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)



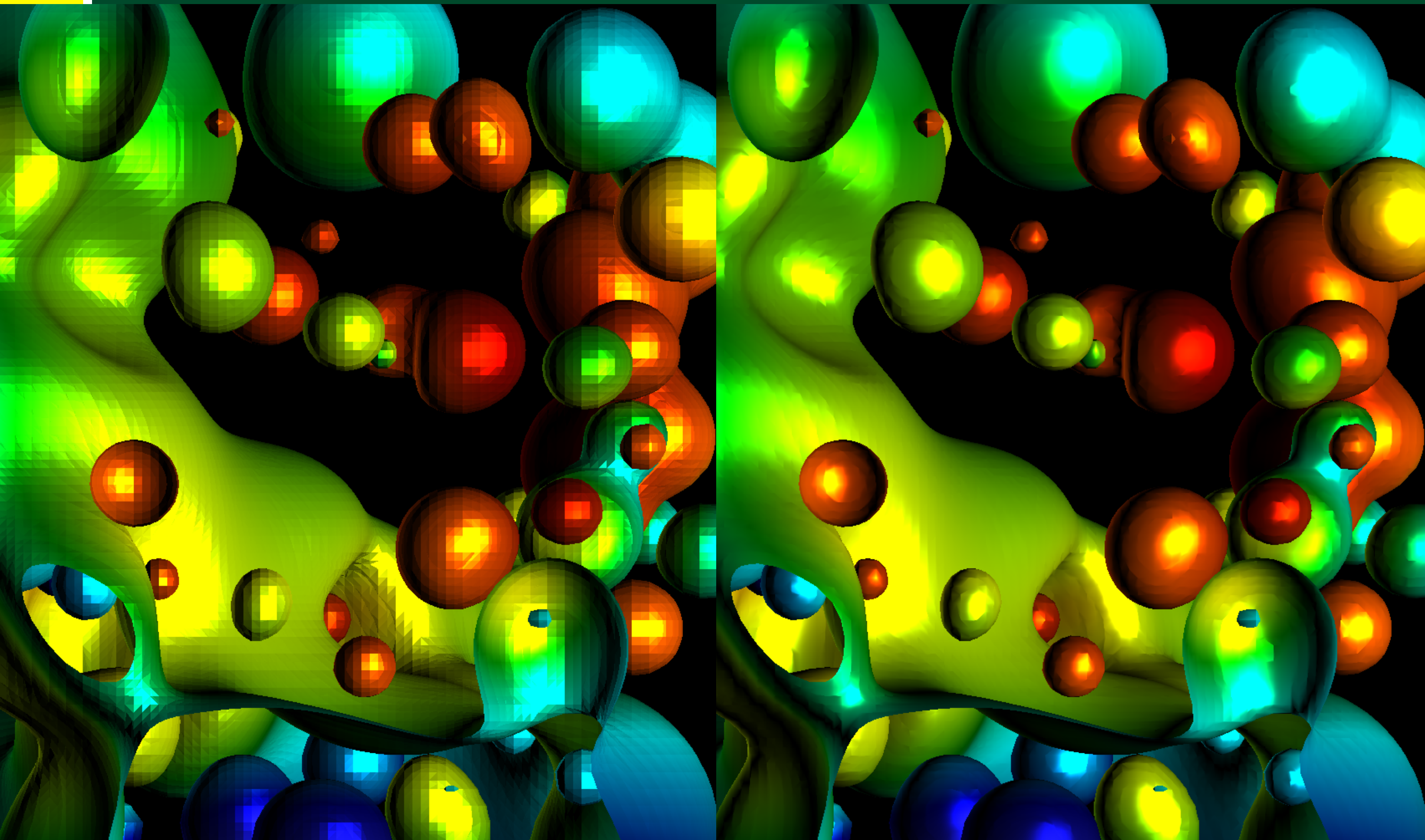
Important:
(C-A)x(B-A) != (B-A)x(C-A)
… we'll worry about this later

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



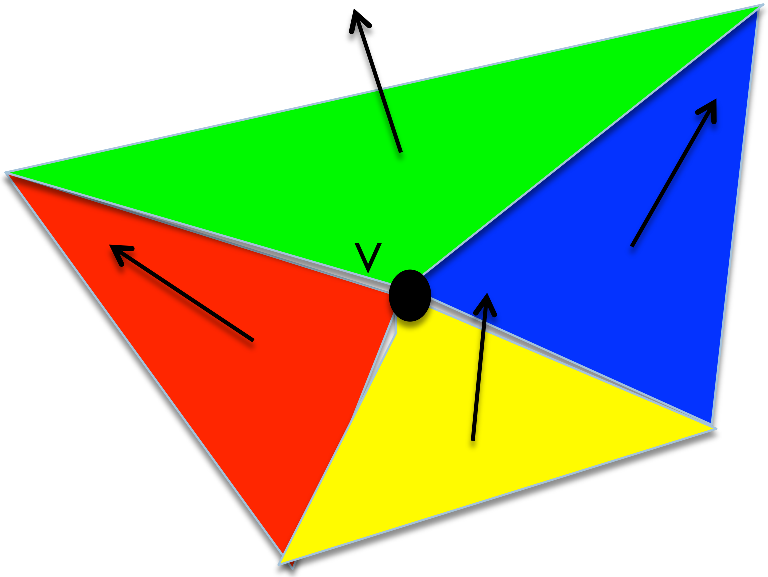
Lighting and Normals

- 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 calculate shading at vertex, then LERP shading
    - How do you calculate normal at a vertex?

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 1F

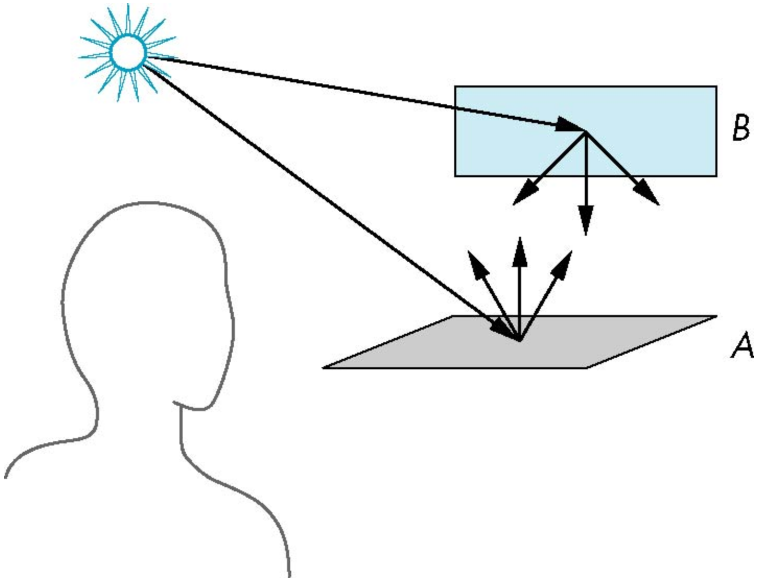
Outline

- Math Basics
- Lighting Basics
- The Phong Model

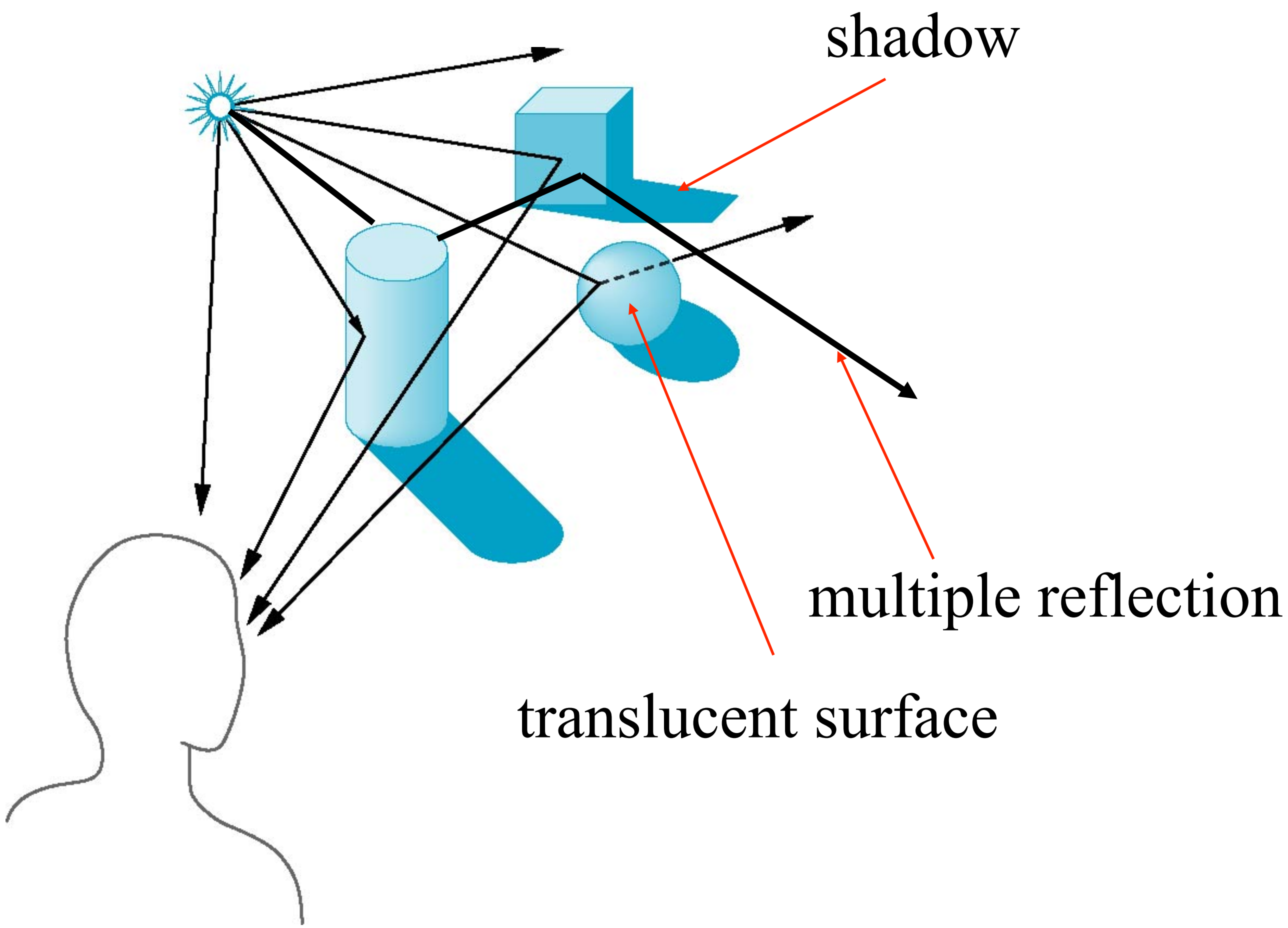
Scattering

- 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 (1/2)

- Local rendering: when rendering one triangle, ignore the effects of other triangles
- Global rendering: when rendering one triangle, consider the effects of other triangles

Local vs Global Rendering (2/2)

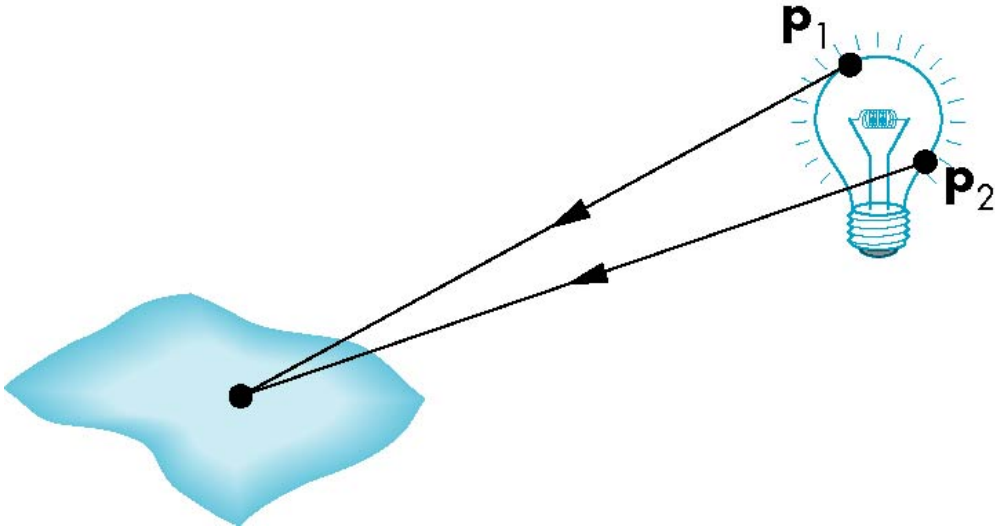
- Correct shading requires a global calculation involving all objects and light sources
  - Incompatible with model which shades each polygon independently (local rendering)
- However, in computer graphics, especially real time graphics, we are happy if things "look right"
  - Many techniques exist for approximating global effects
    - I.e., do local rendering, but bring in other knowledge to make it look like global rendering

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



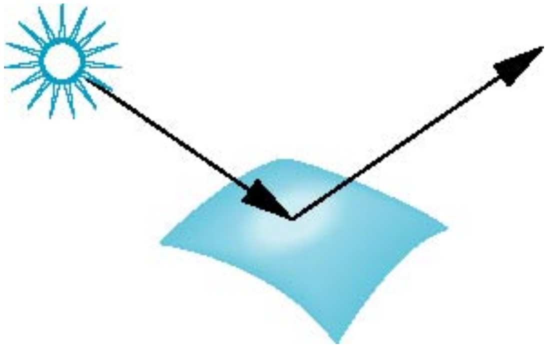
Simple Light Sources

- Point source
  - Model with position and color
  - Distant source = infinite distance away (parallel)
- Spotlight
  - Restrict light from ideal point source

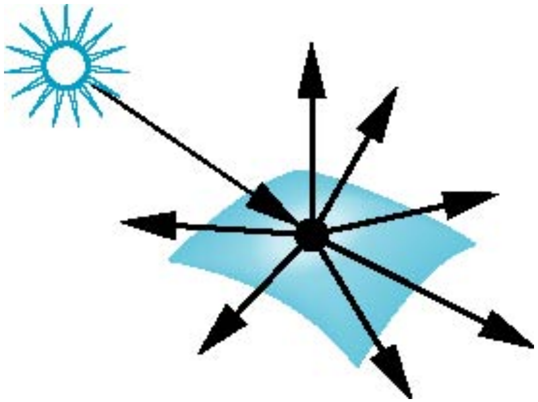
- (We will do point sources for 1F … and this class)

Surface Types

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



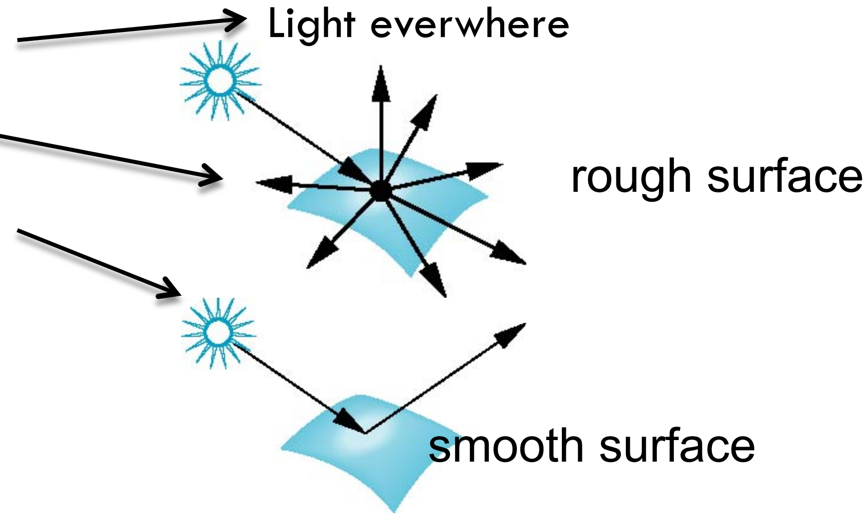
smooth surface



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



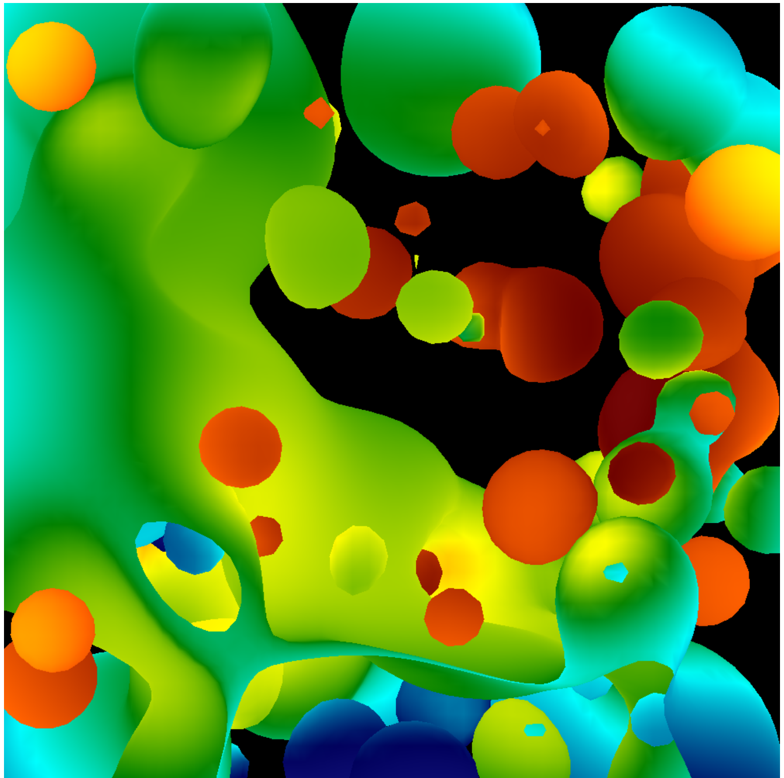
<u>Our game plan:</u>
Calculate all 3 and combine them.

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

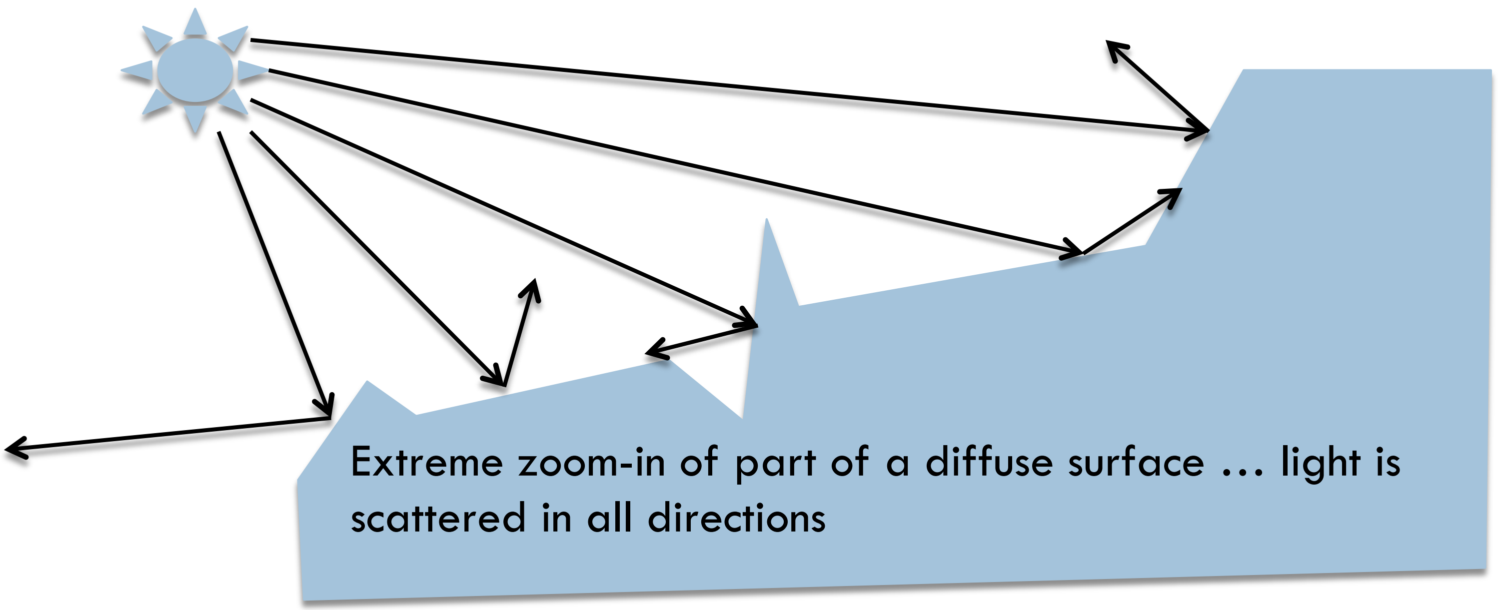
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces



Surface lit with ambient lighting only

Lambertian Surface

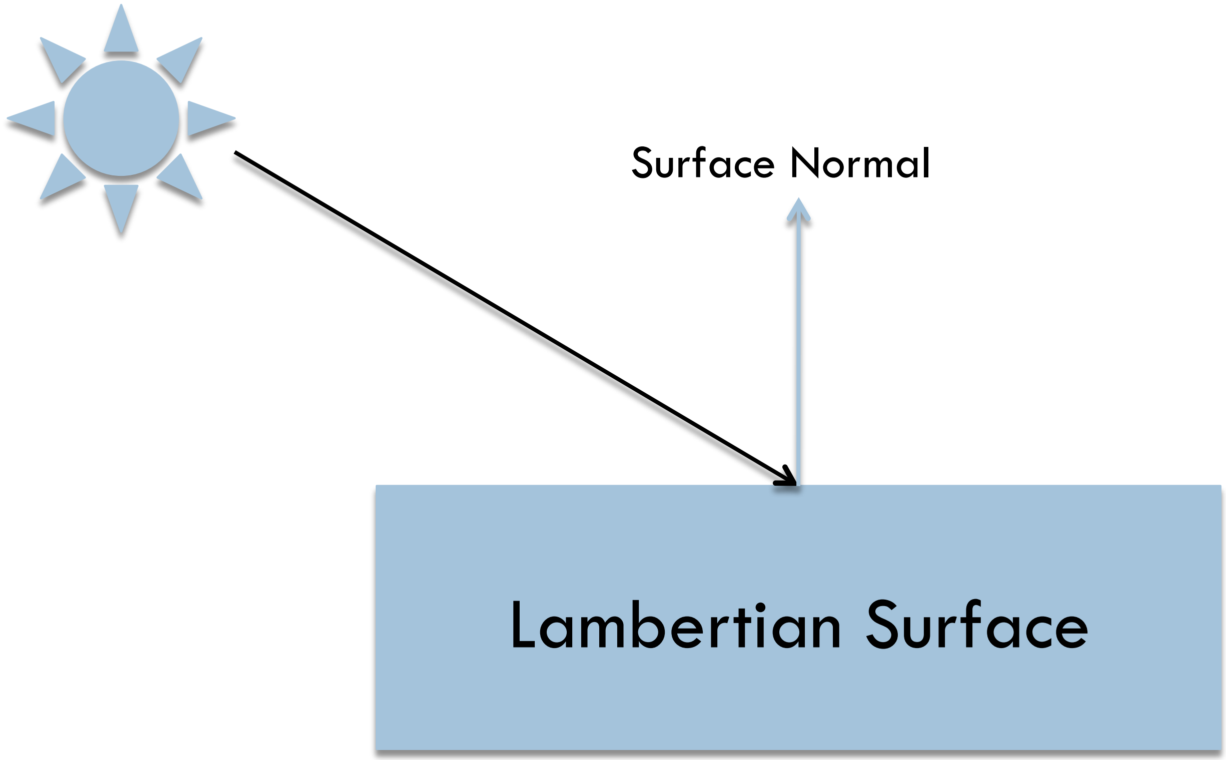
- Perfectly diffuse reflector
- Light scattered <u>equally</u> in all directions



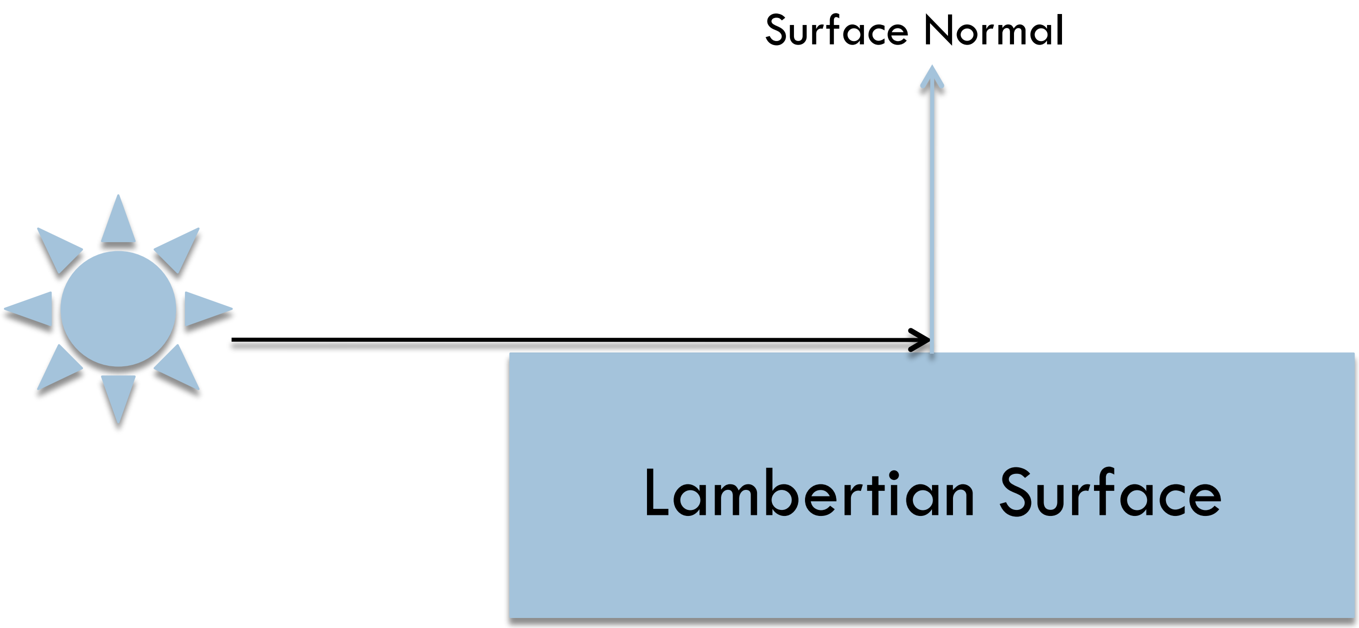
(this image shows 5 of the directions)

Slide inspired by Ed Angel Computer Graphics Book

Diffuse Lighting

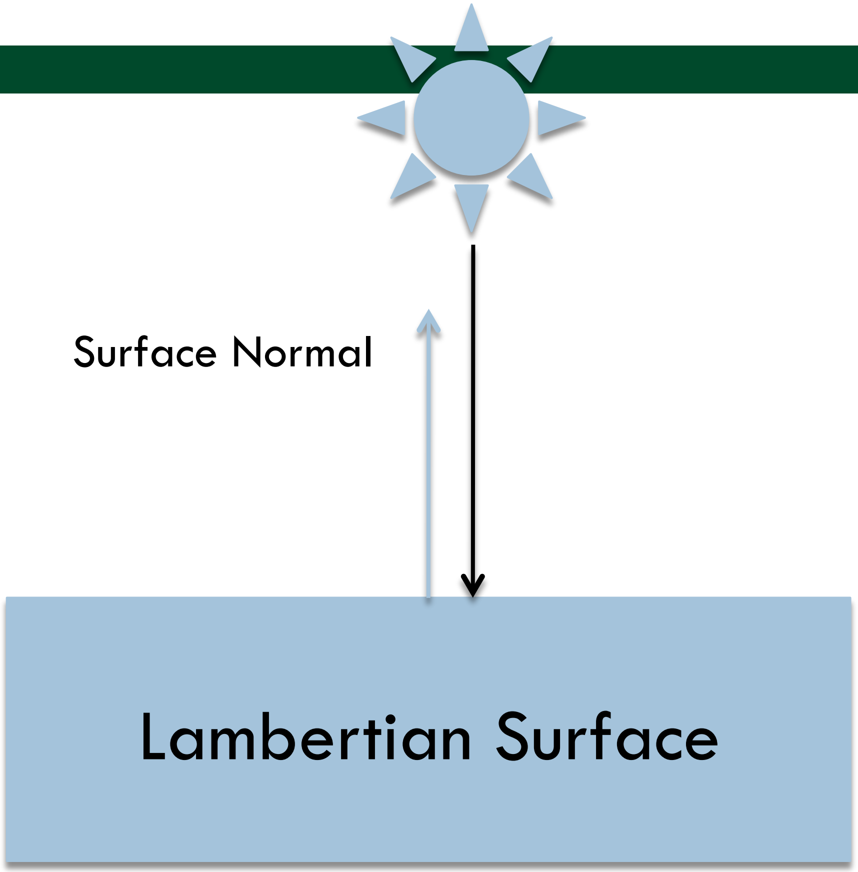


Diffuse Lighting



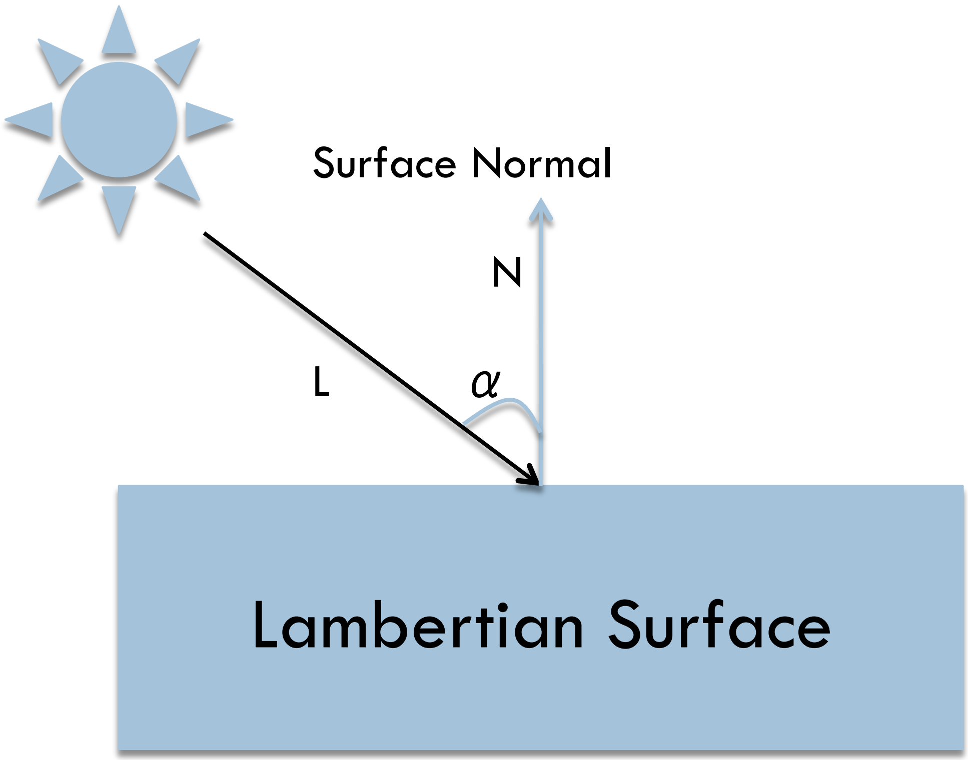
No light reflects off the (top) surface
(Light direction and surface normal are perpendicular)

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?

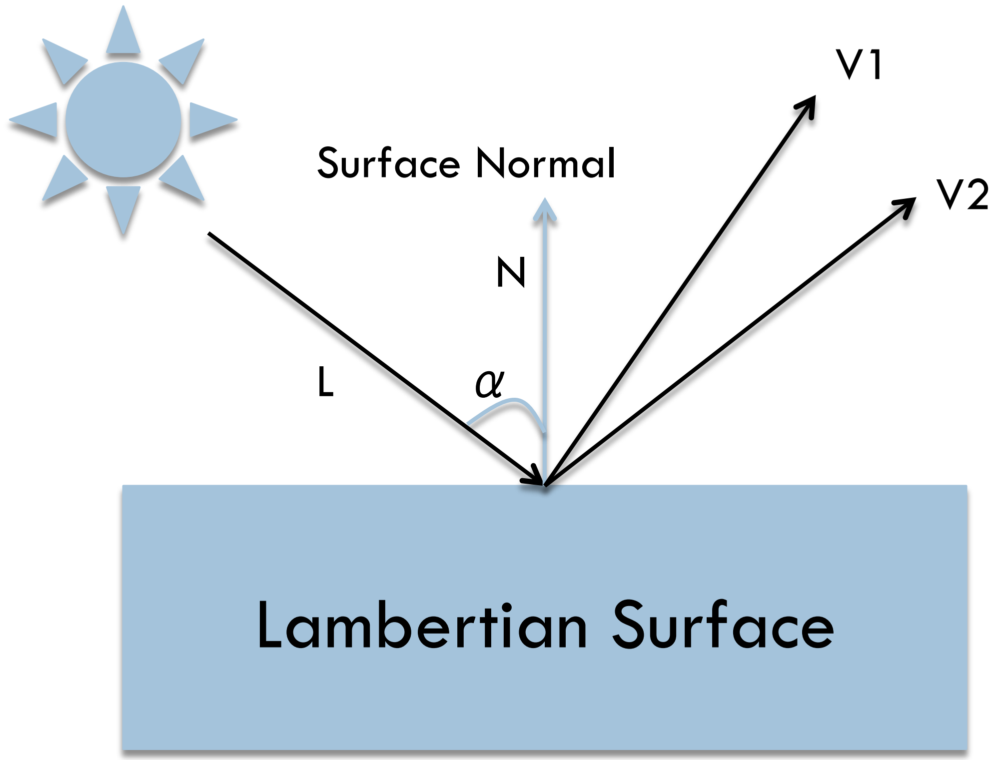
<u>A: cos(α)</u>

And note that:

cos(0) = 1

cos(90) = 0

Diffuse Lighting



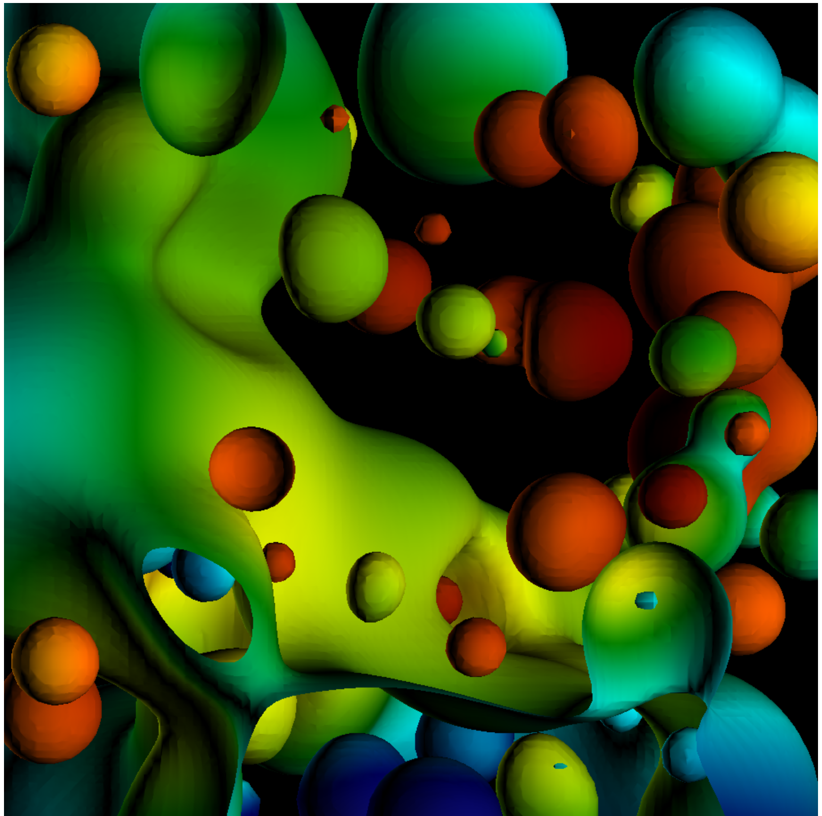
How much light makes it to viewer V1? Viewer V2?

A: cos(α) for both

Lambertian surfaces reflect light equally in all directions

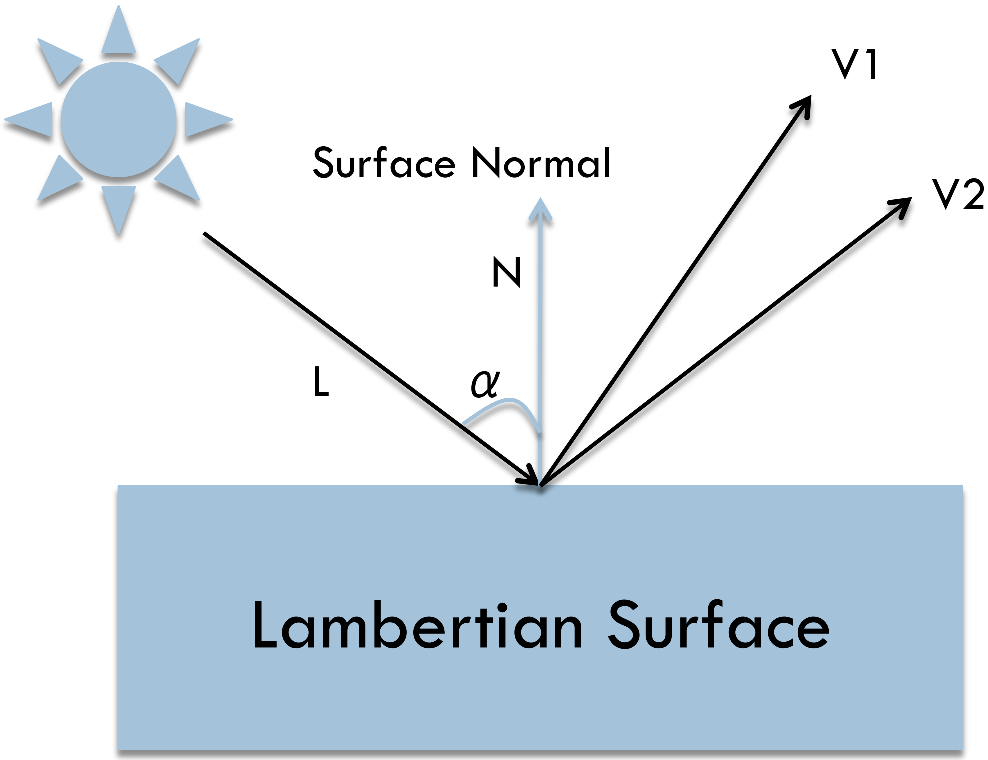
Diffuse Lighting

- 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



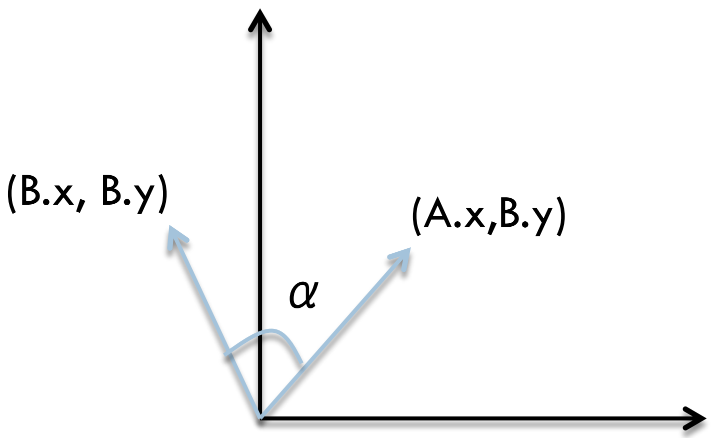
How much light makes it to viewer V1? Viewer V2?

A: cos(α) for both

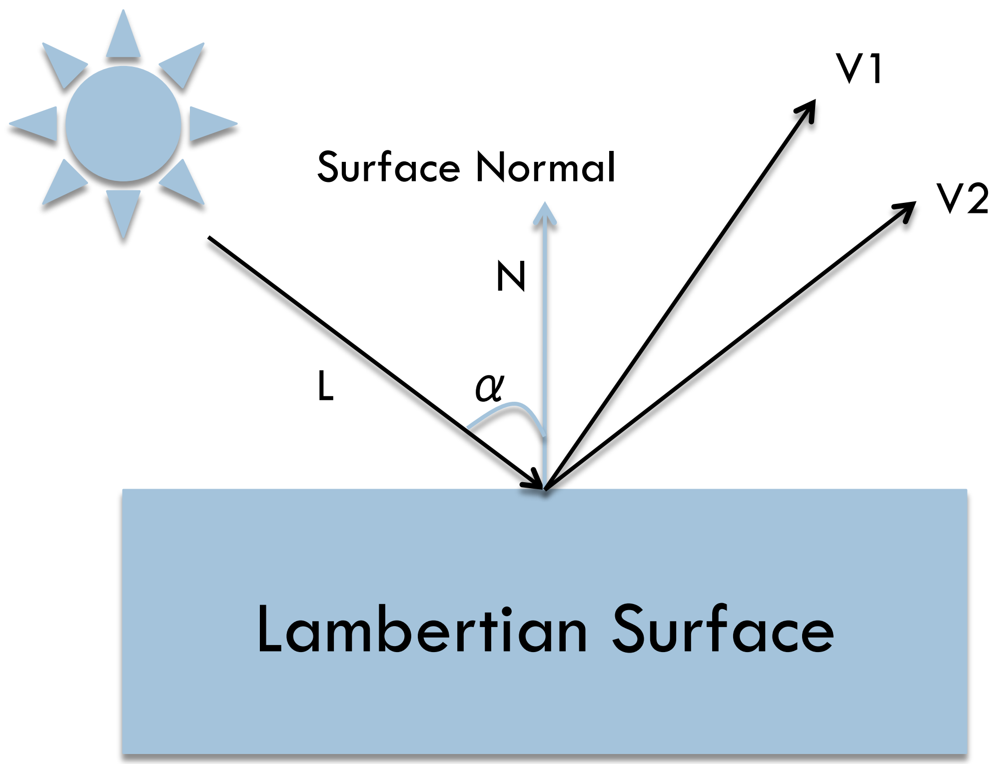
Lambertian surfaces reflect light equally in all directions

What is a dot product?

- A·B = A.x*B.x + A.y*B.y
- Physical interpretation:
  - A·B = cos(α)/(||A||*||B||)

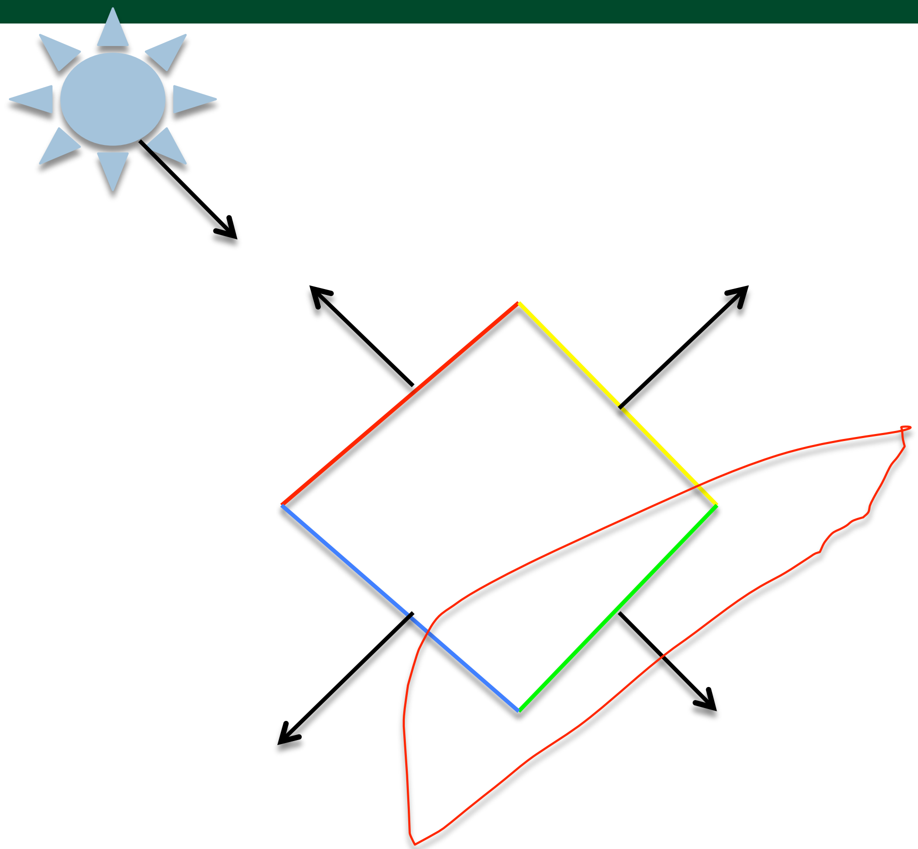


Diffuse Lighting

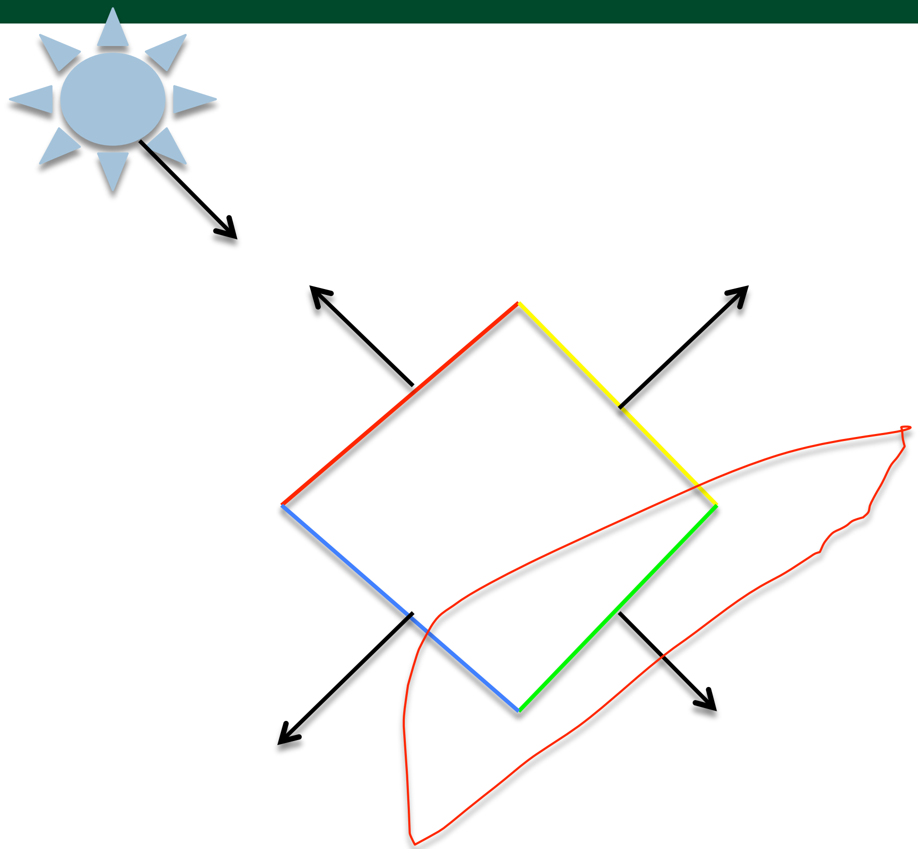


You can calculate the diffuse contribution by taking the dot product of L and N,
Since $L \cdot N = \cos(\alpha)$
(assuming L and N are normalized)

What about cases where L·N < 0?



What about cases where L·N < 0?



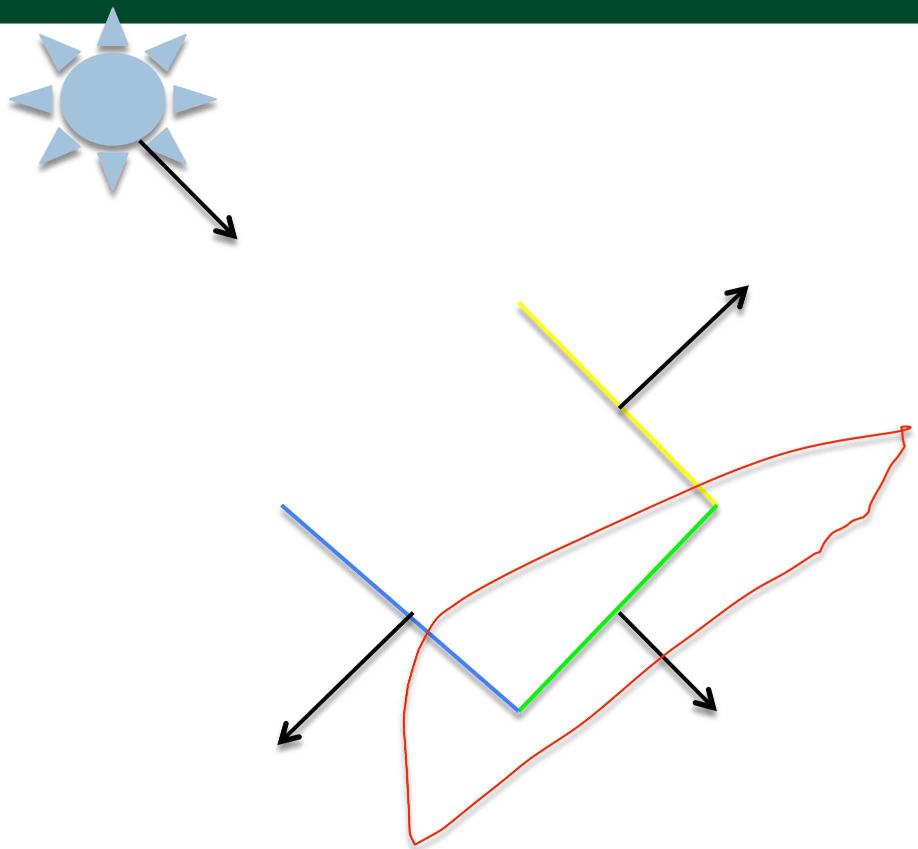
L·N = -1

Non-sensical … takes away light?

Common solution:

Diffuse light = max(0, L·N)

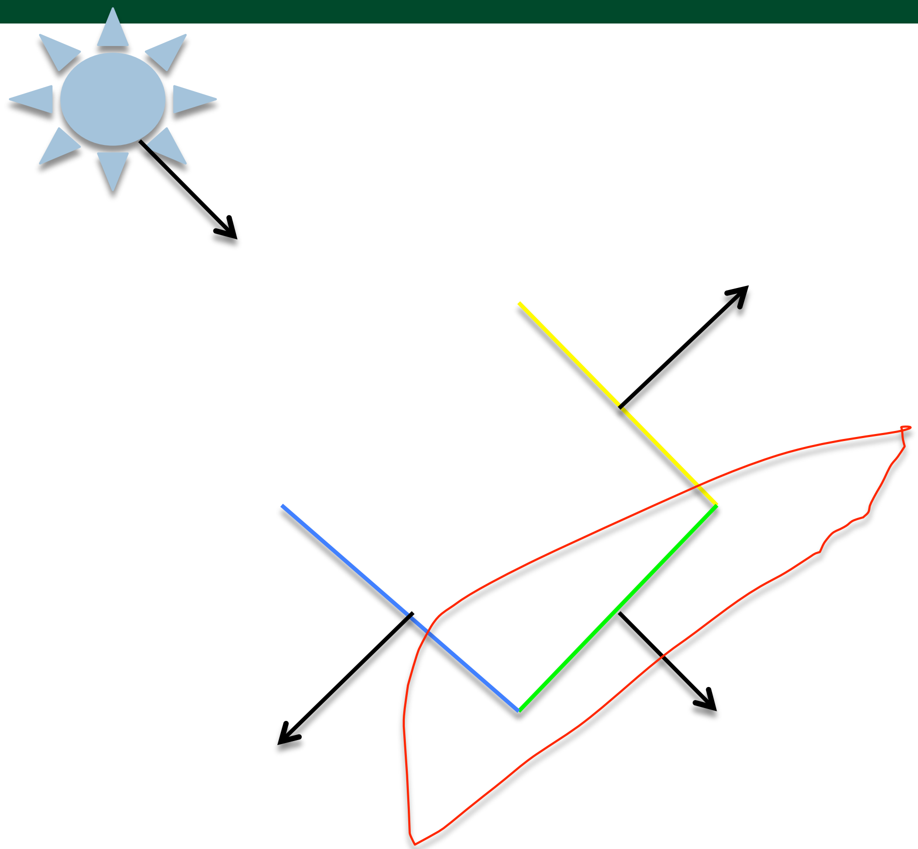
But wait…



If you have an open surface, then there is a "back face".

The back face has the opposite normal.

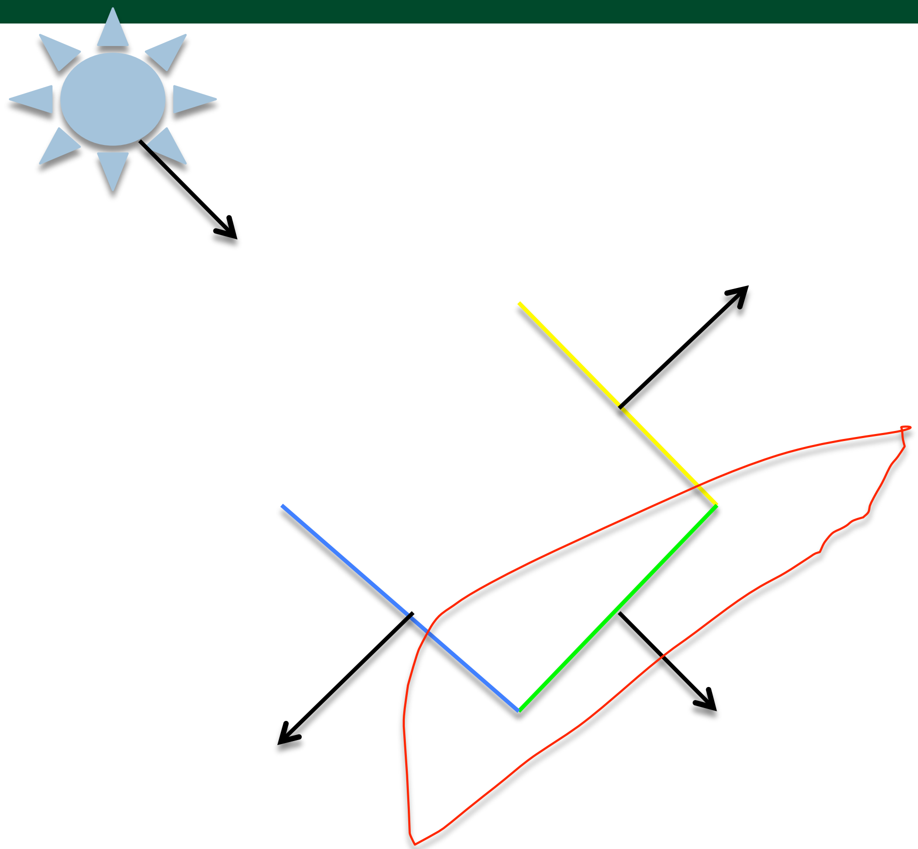
But wait…



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…

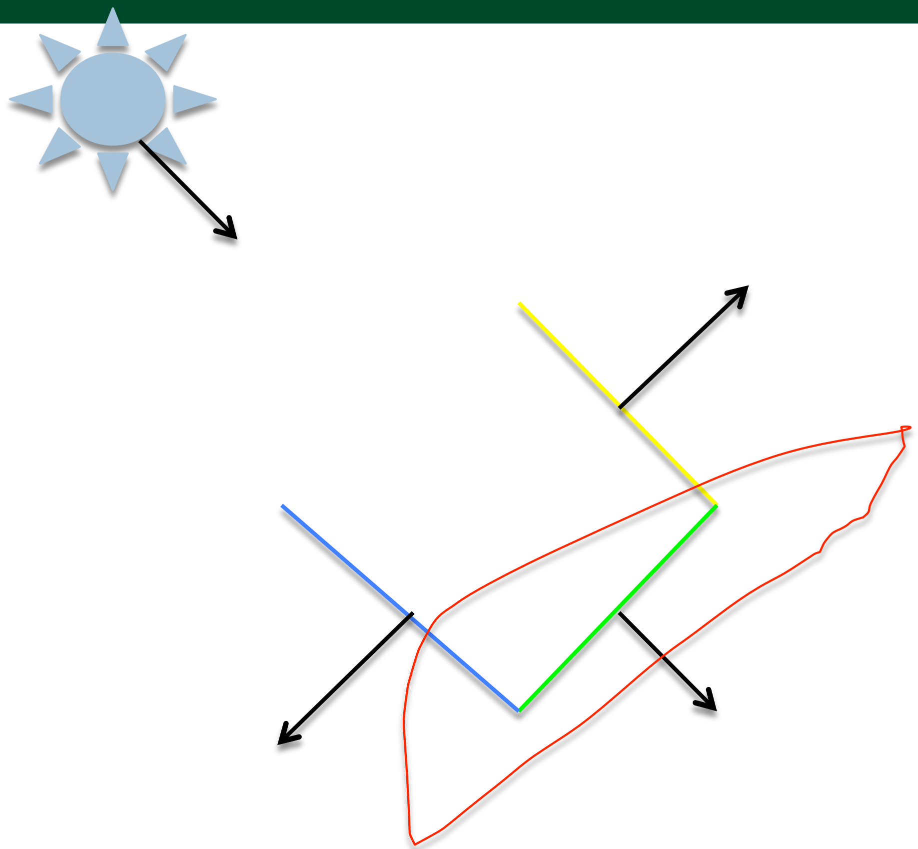


If you have an open surface, then there is a “back face”.
The back face has the opposite normal.

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?

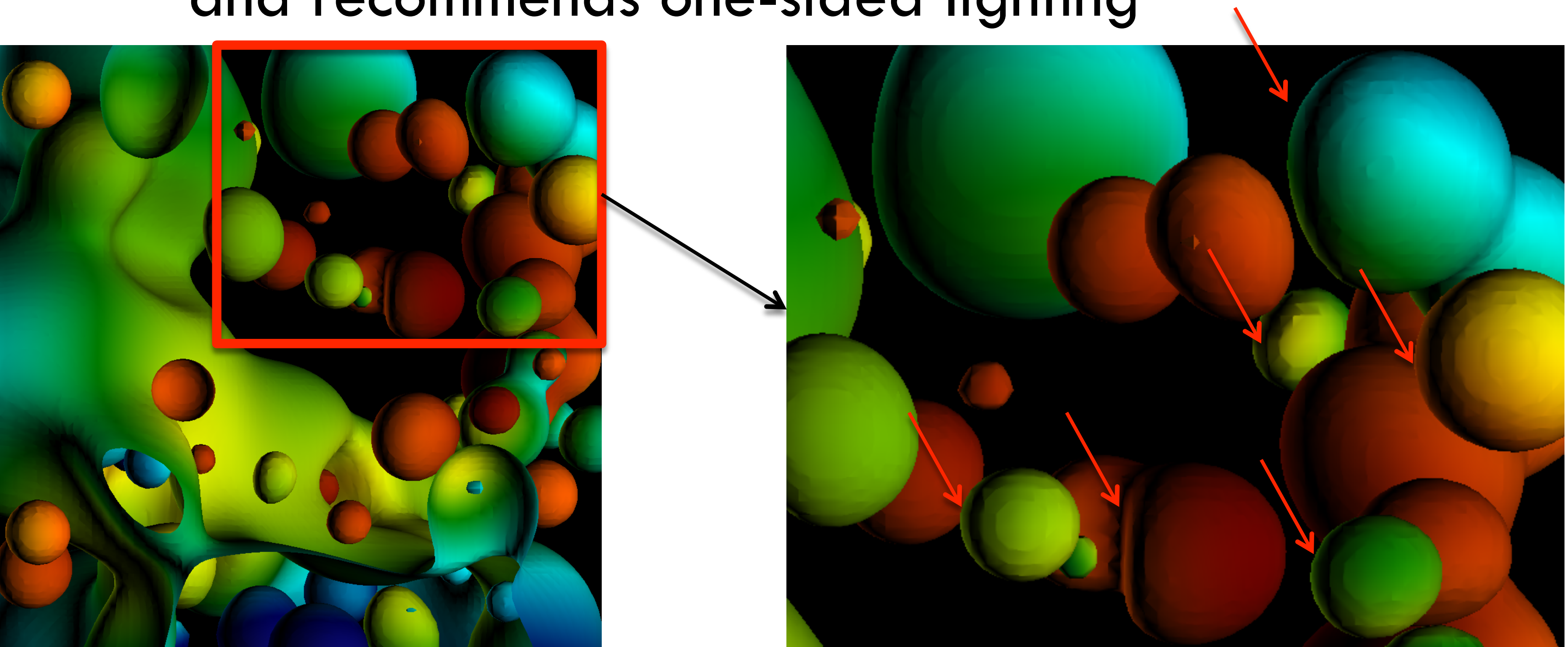
Idea #1: encode all triangles twice, with different normals
Idea #2: modify diffuse lighting model

Diffuse light = abs(L·N)

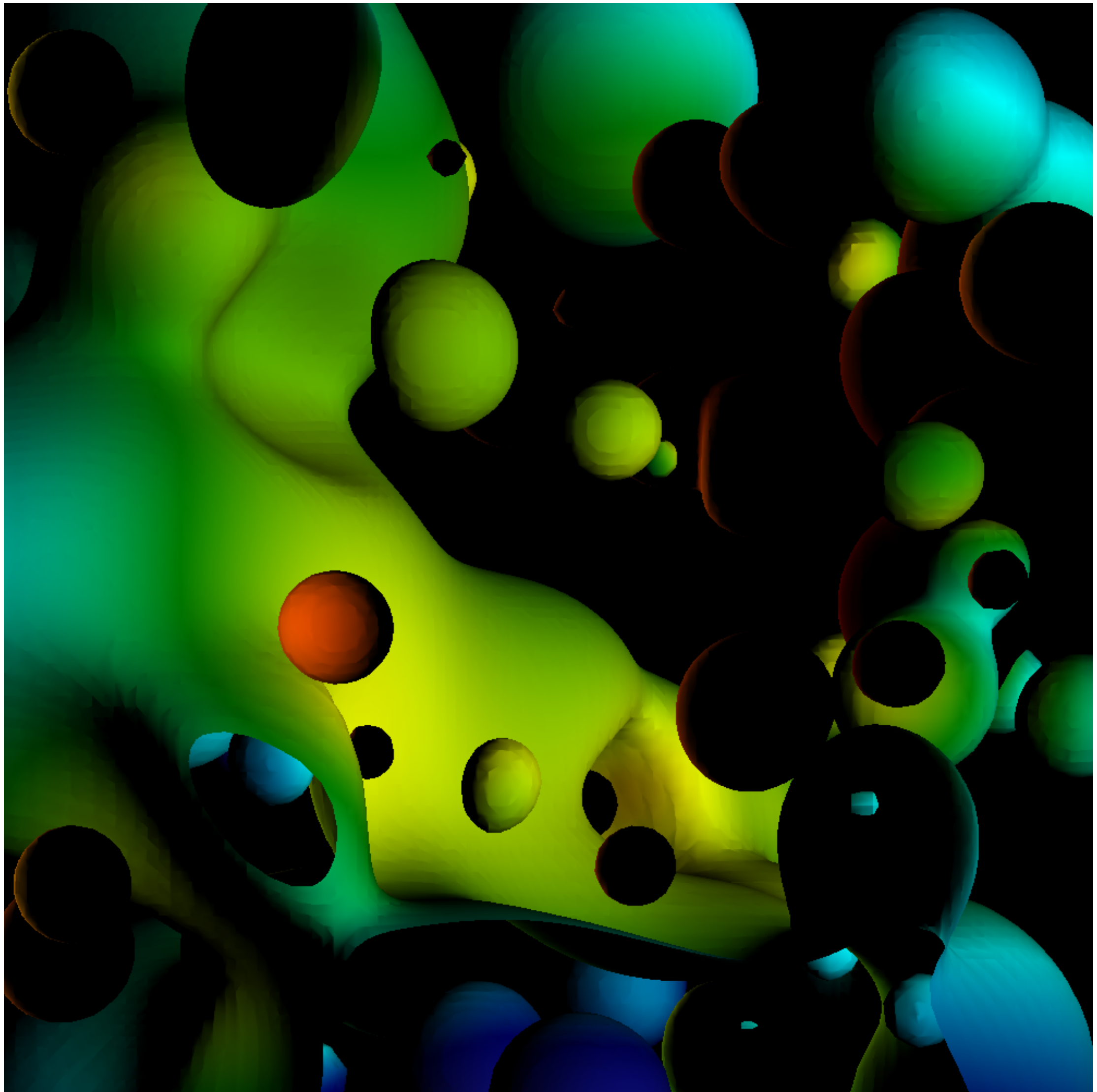
This is called two-sided lighting

Two-sided lighting

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

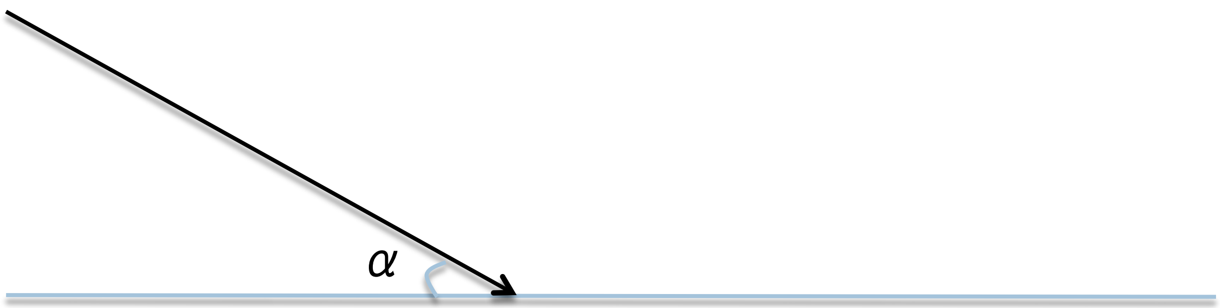


One-sided lighting with open surfaces is disappointing



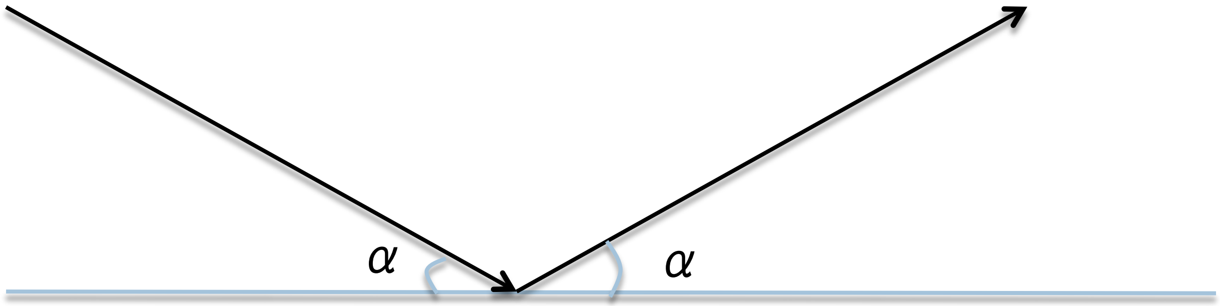
The most valuable thing I learned in Freshman Physics

- "angle in = angle out"

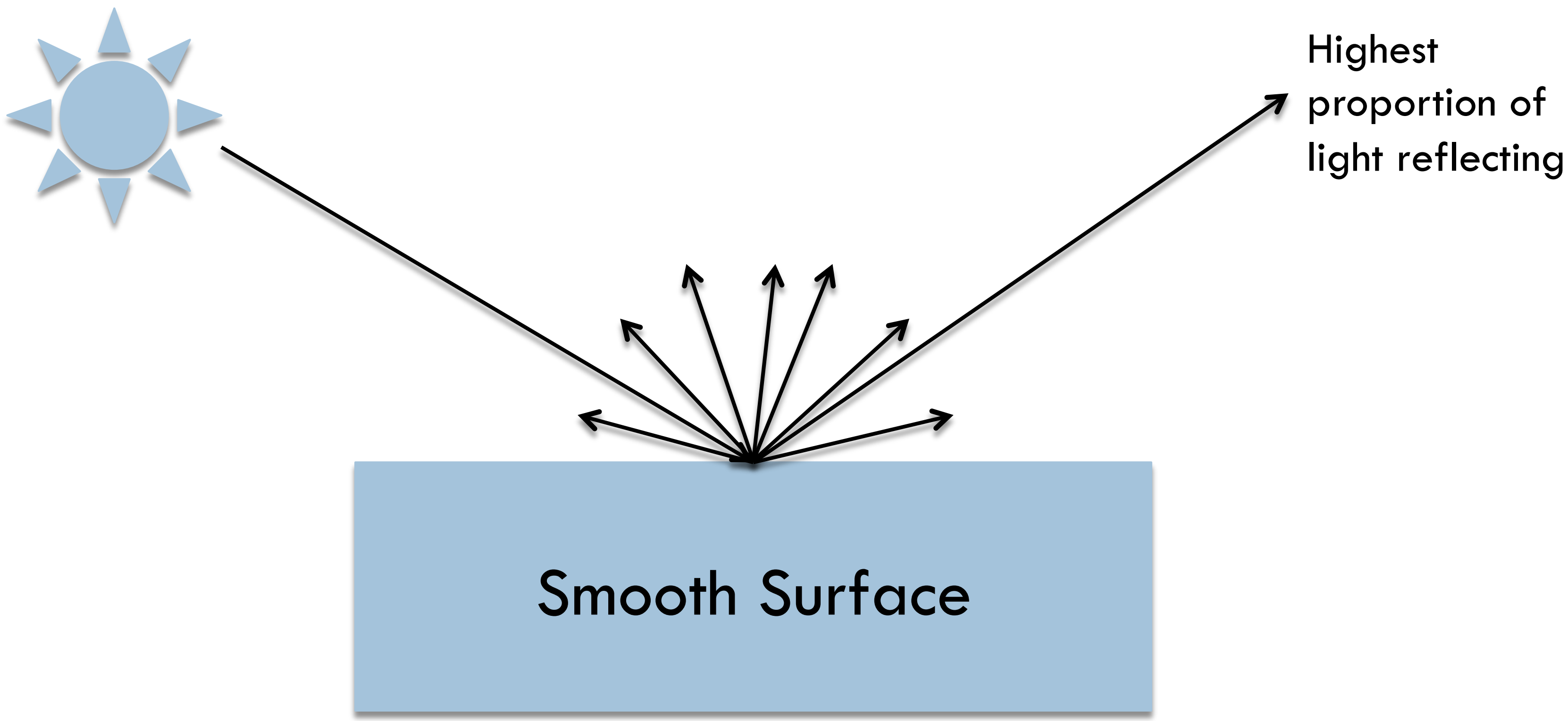


The most valuable thing I learned in Freshman Physics

- "angle in = angle out"

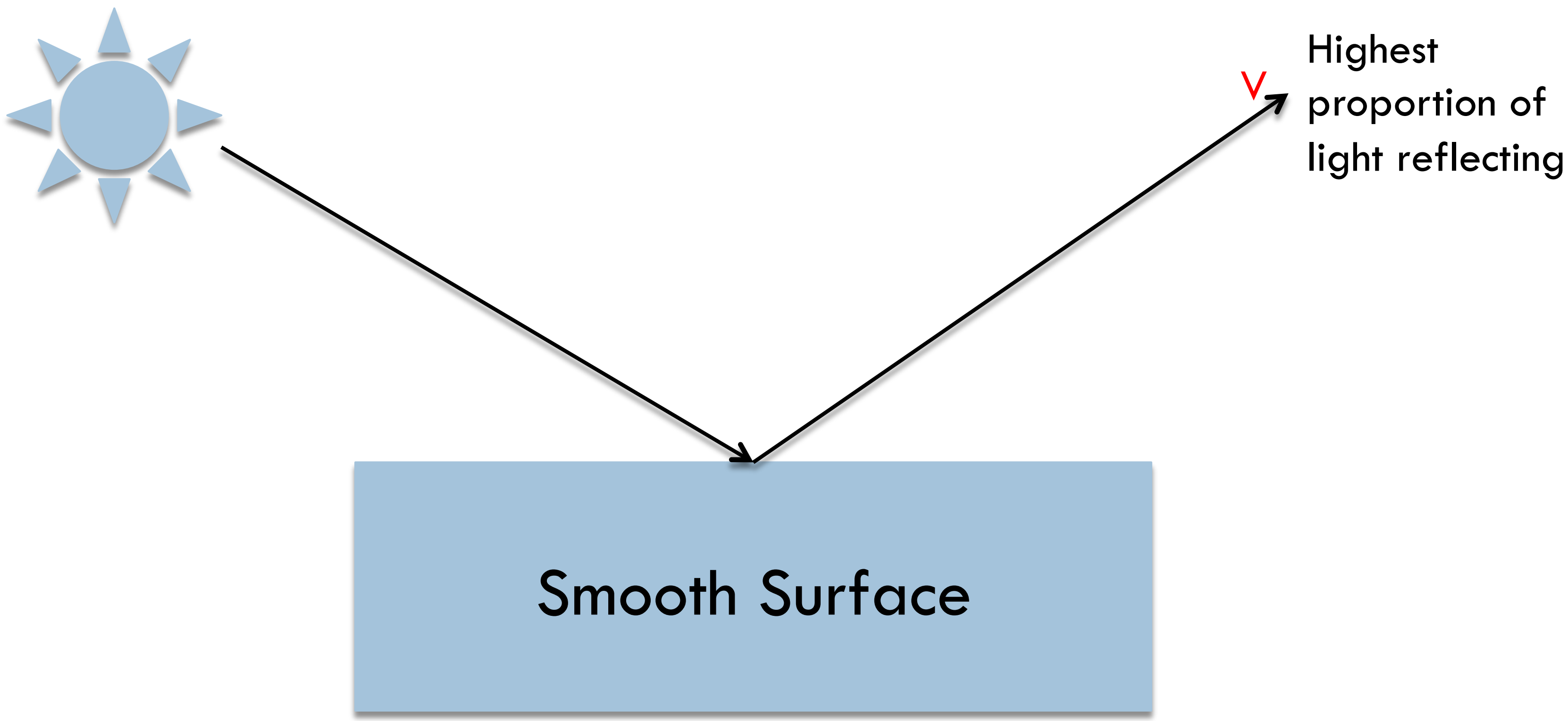


Specular Lighting



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?

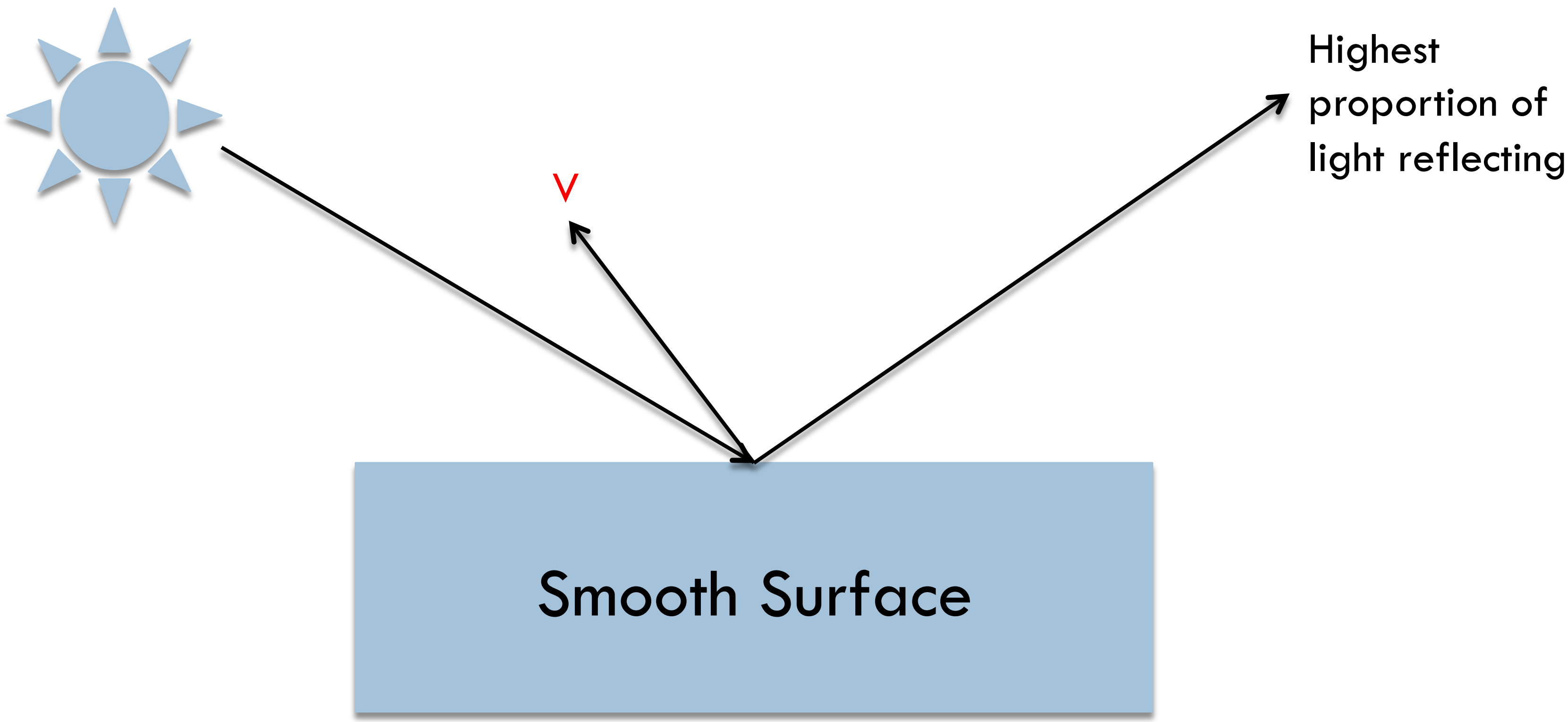


Consider V located along reflection ray.

Answer: most possible

Call this "1"

How much light reflects with specular lighting?

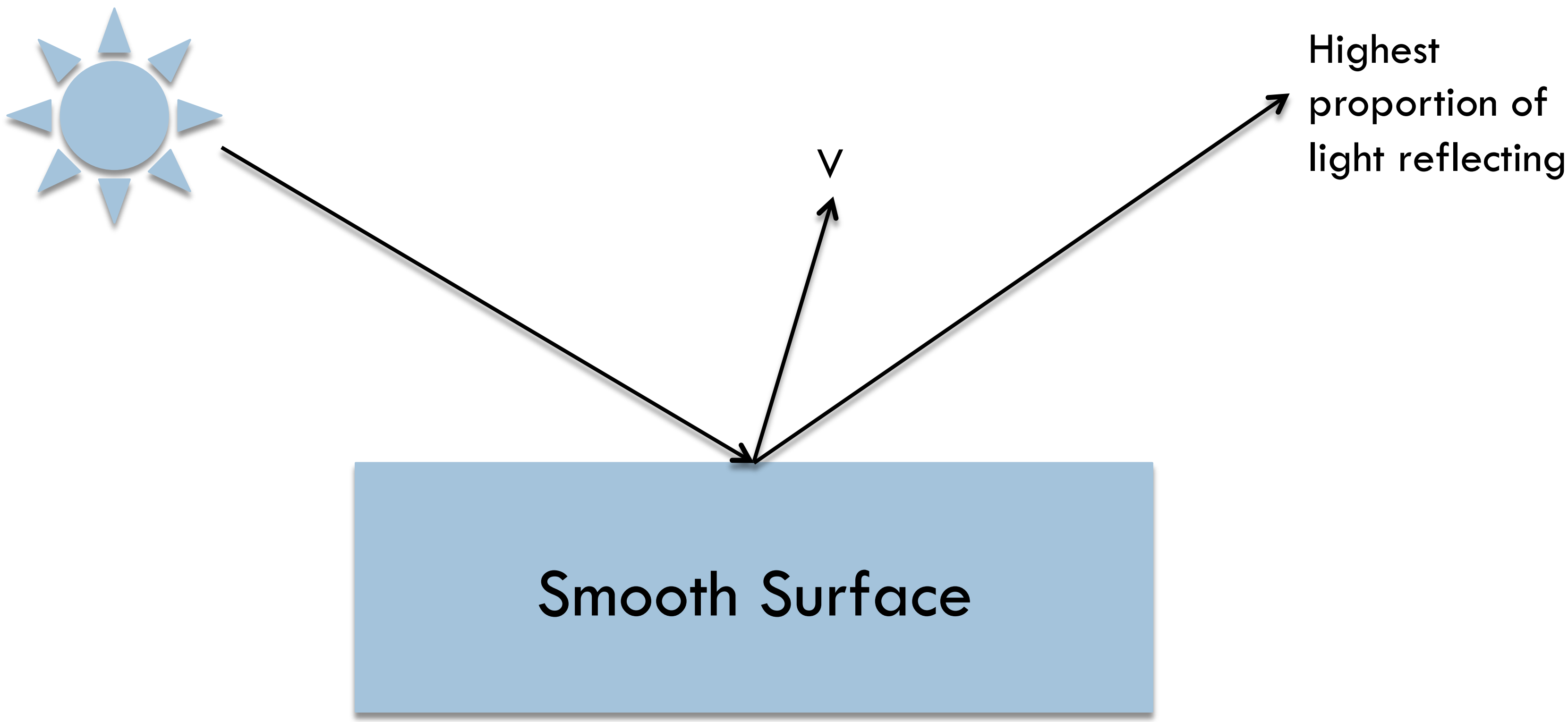


Consider V located along perpendicular ray.

<u>Answer</u>: none of it

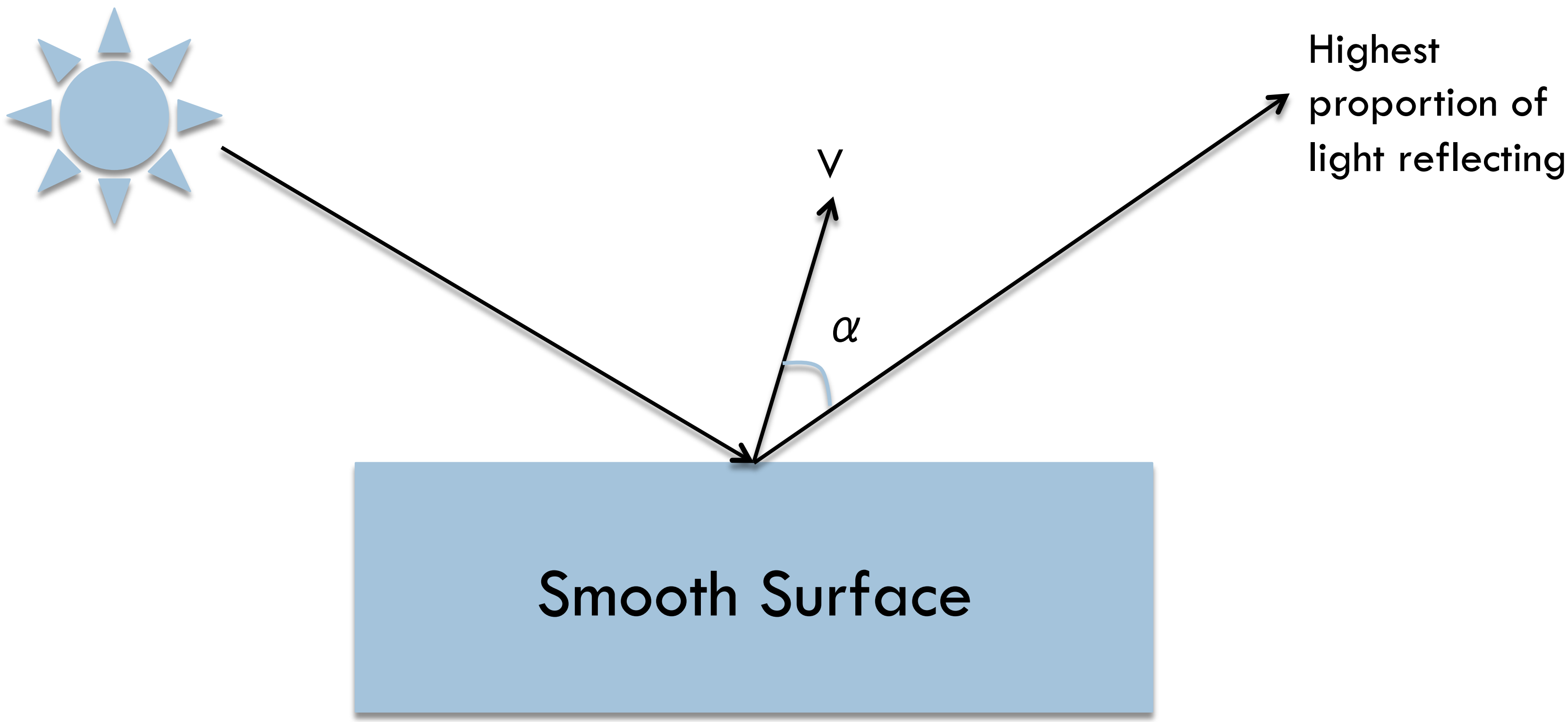
Call this "0"

How much light reflects with specular lighting?



How much light gets to point V?

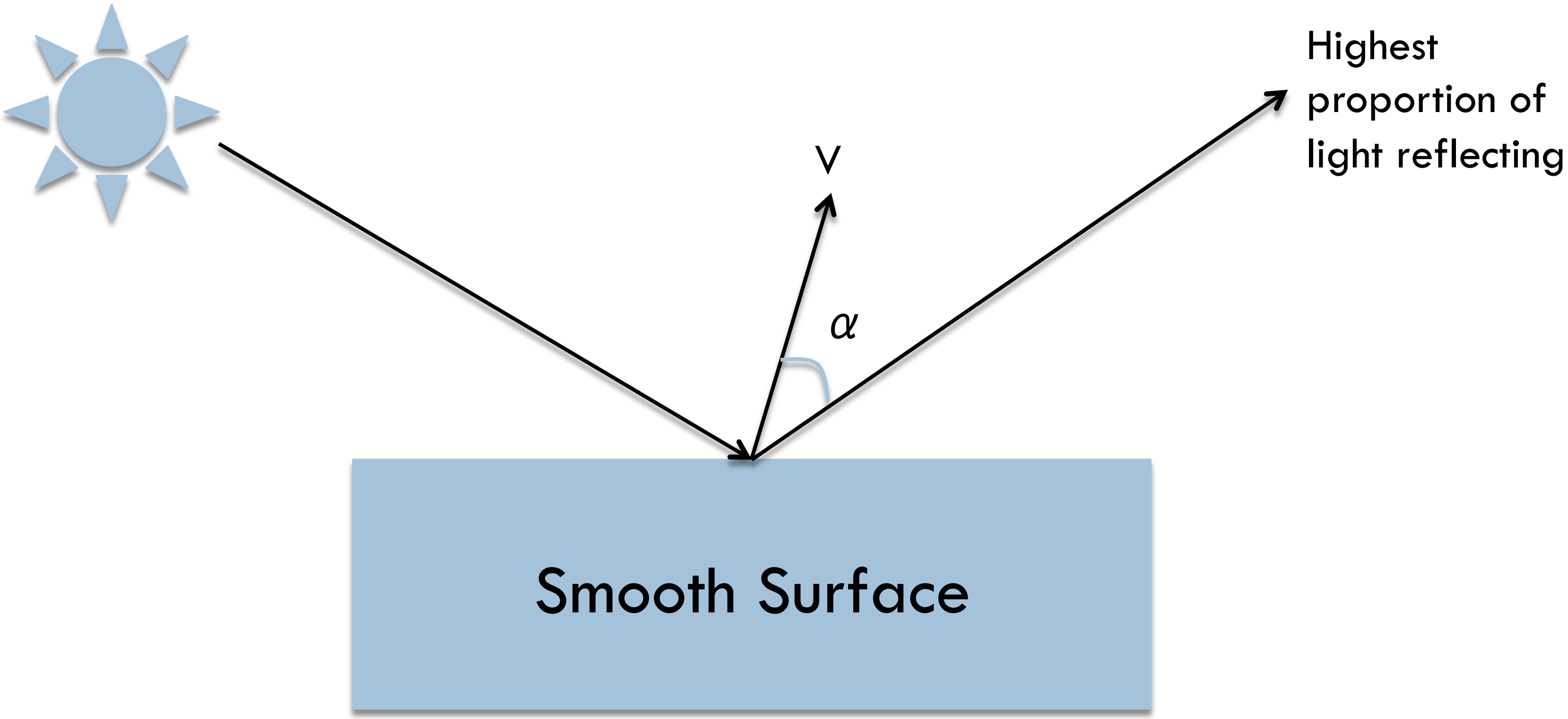
How much light reflects with specular lighting?



How much light gets to point V?

A: proportional to cos(α)

How much light reflects with specular lighting?



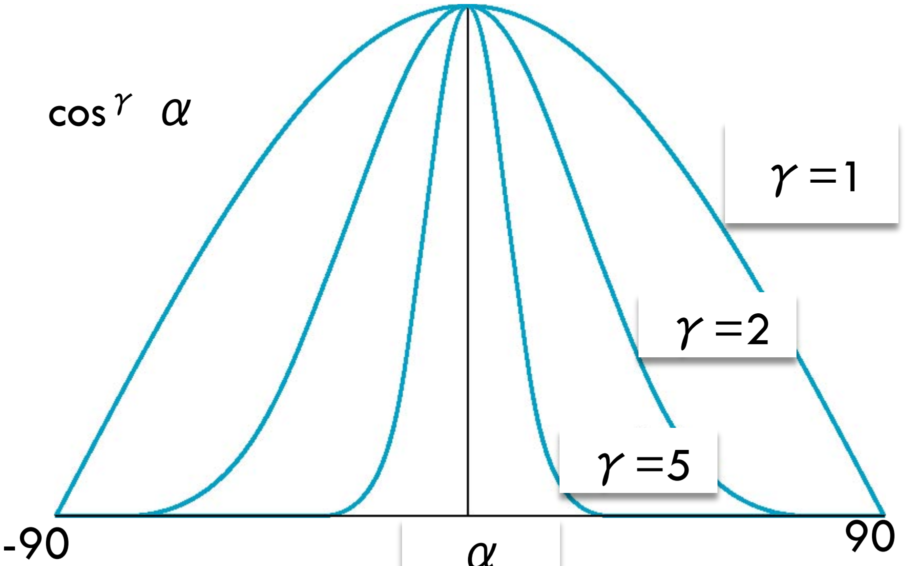
How much light gets to point V?

A: proportional to cos(α)

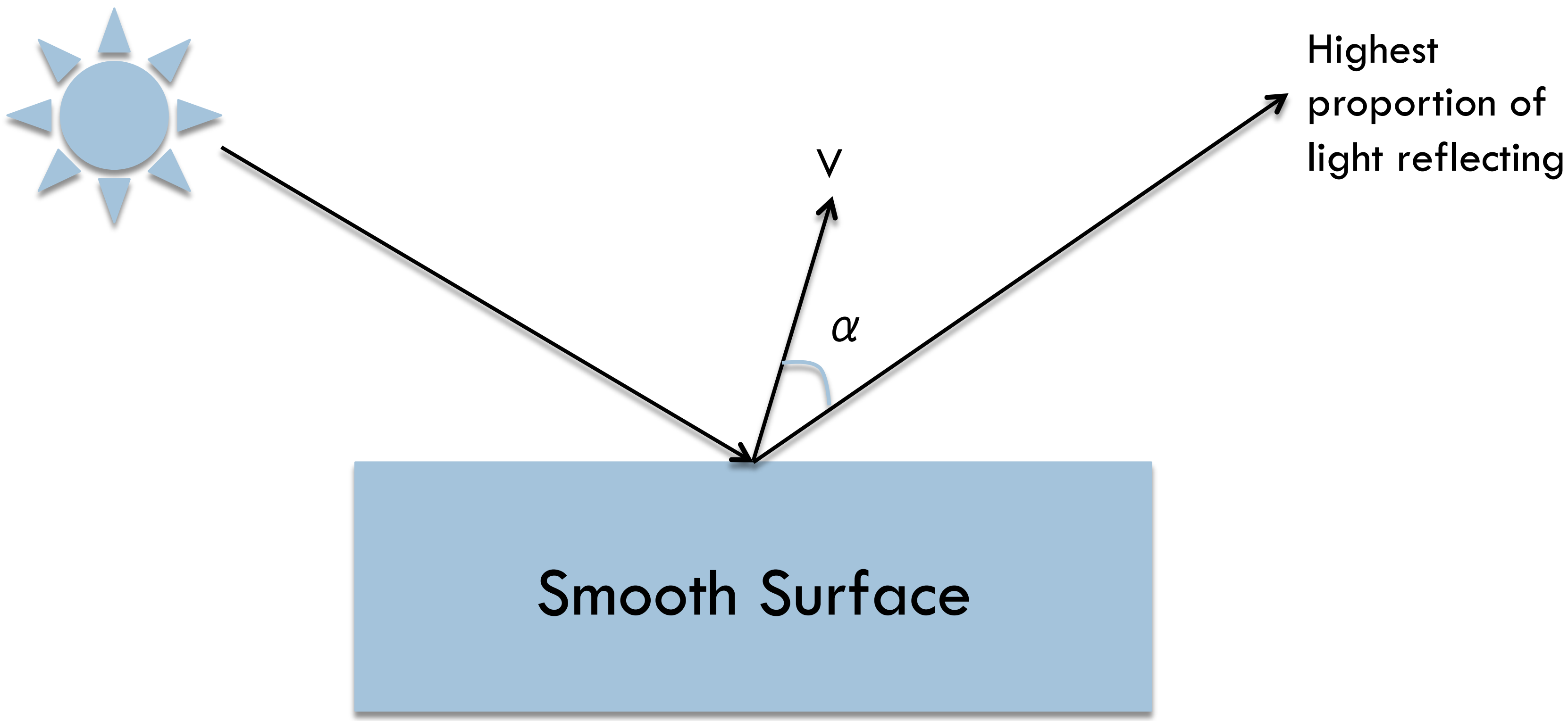
(Shininess strength) * cos(α) ^ (shininess coefficient)

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?

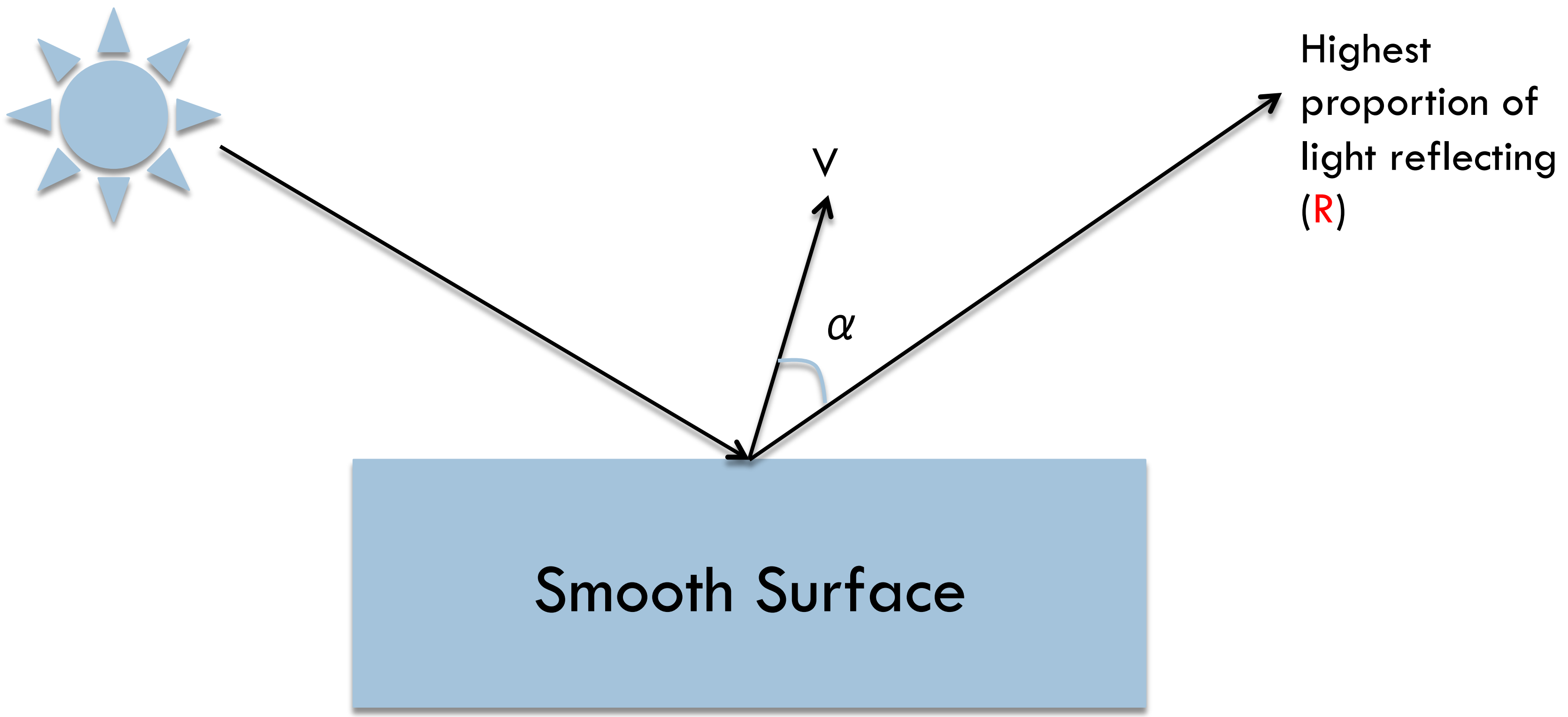


How much light gets to point V?

A: proportional to cos(α)

(Shininess strength) * cos(α) ^ (shininess coefficient)

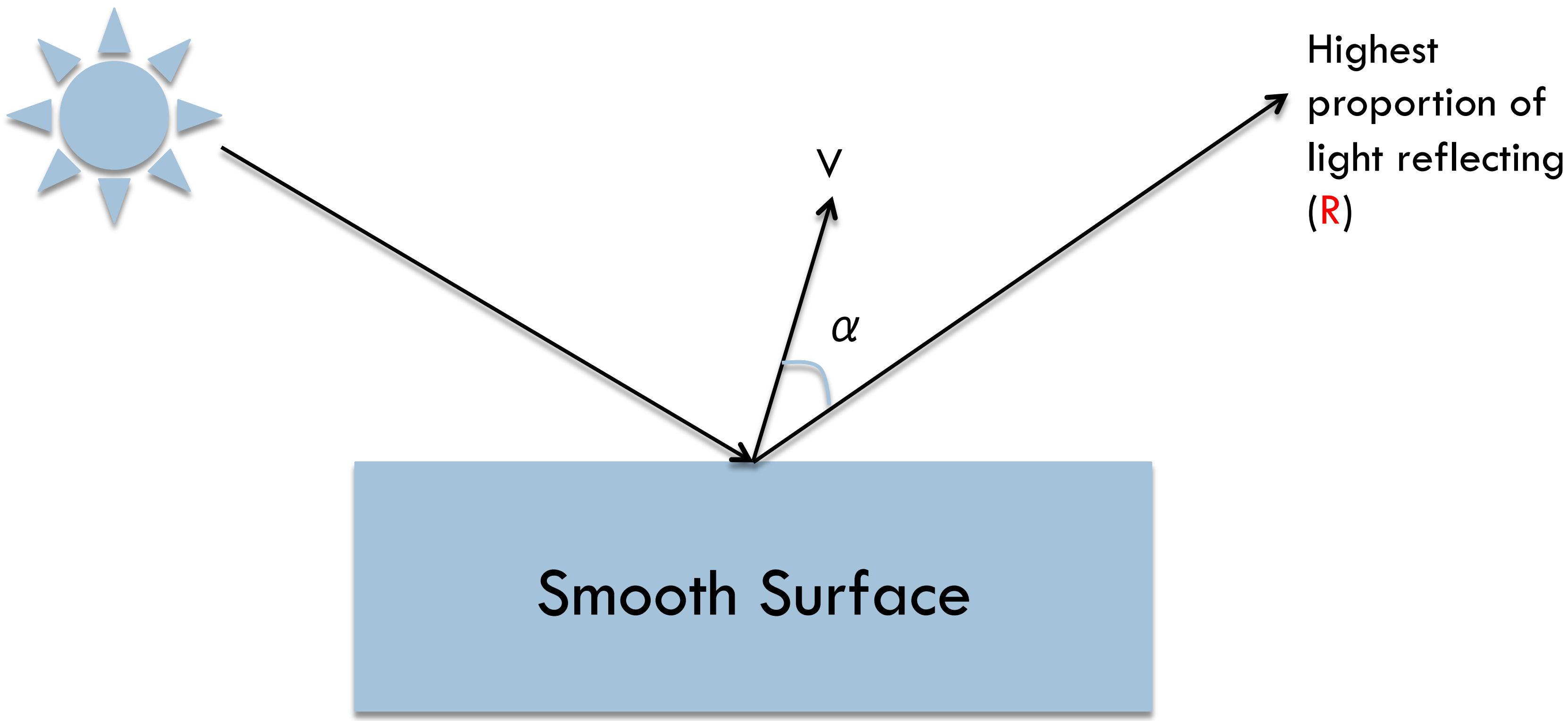
How much light reflects with specular lighting?



Great!

We know that cos(α) is V·R (provided V & R are normalized).

How much light reflects with specular lighting?



Great!

We know that cos(α) is V·R (provided V & R are normalized).

But what is R?

It is a formula: R = 2*(L·N)*N - L

Two-sided lighting

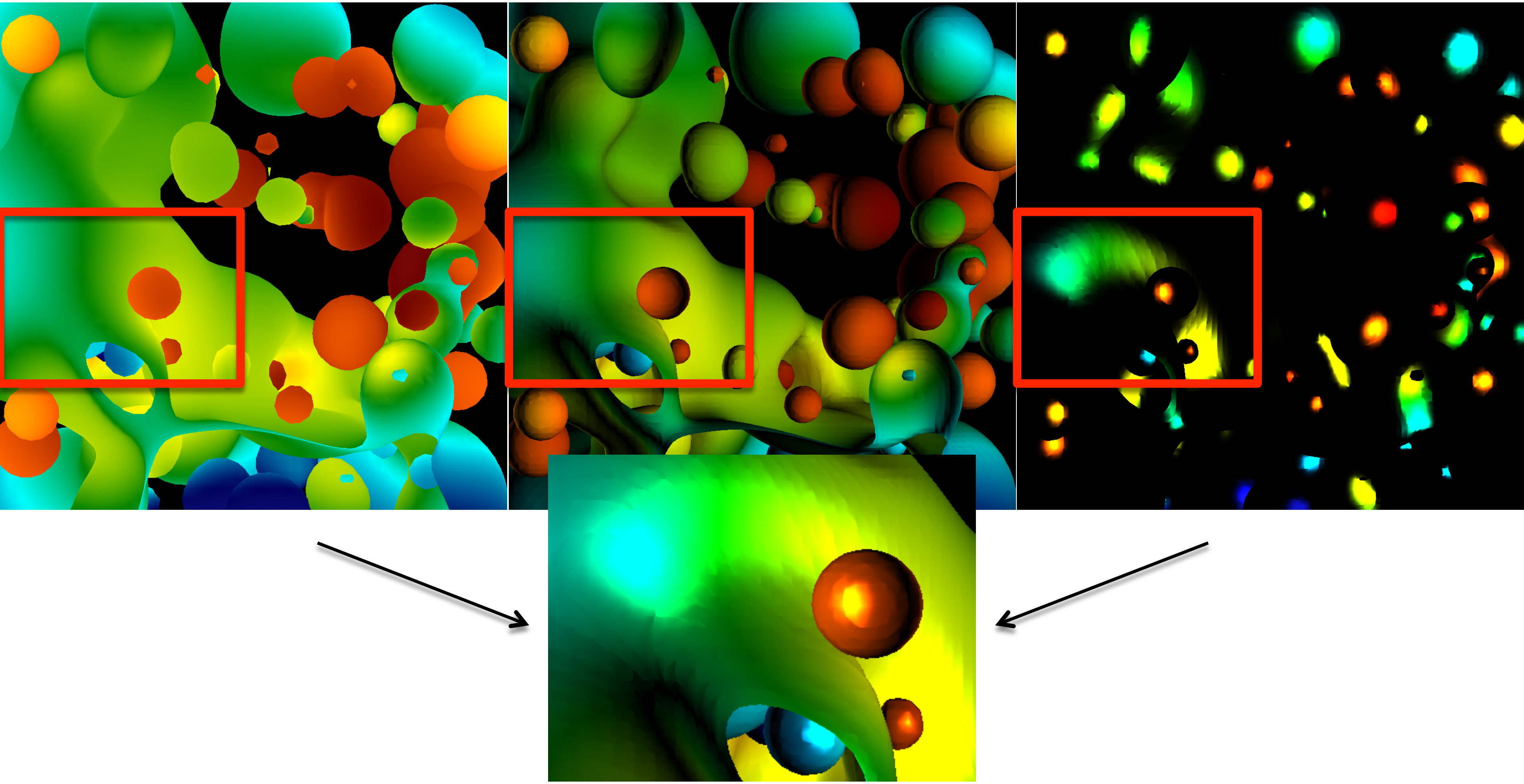
- For specular lighting, we will use one-sided lighting for project 1F
  - It just looks better
  - Diffuse: abs(L·N)
  - Specular: max(0, S*(R·V)$^{\gamma}$)

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)
  - Will have to calculate viewDirection for each pixel!

Specular Term of Phong Model

- Specular part of Phong: K_s*Specular
- and Specular is: (Shininess strength) * cos(α) ^ (shininess coefficient)
- Putting it all together would be:
  - K_s * (Shininess strength) * cos(α) ^ (shininess coefficient)
- But now we have two multipliers, K_s and (Shininess Strength). Not needed.
- So: just use one. Drop Shininess Strength and only use K_s
  - K_s * cos(α) ^ (shininess coefficient)

Lighting parameters

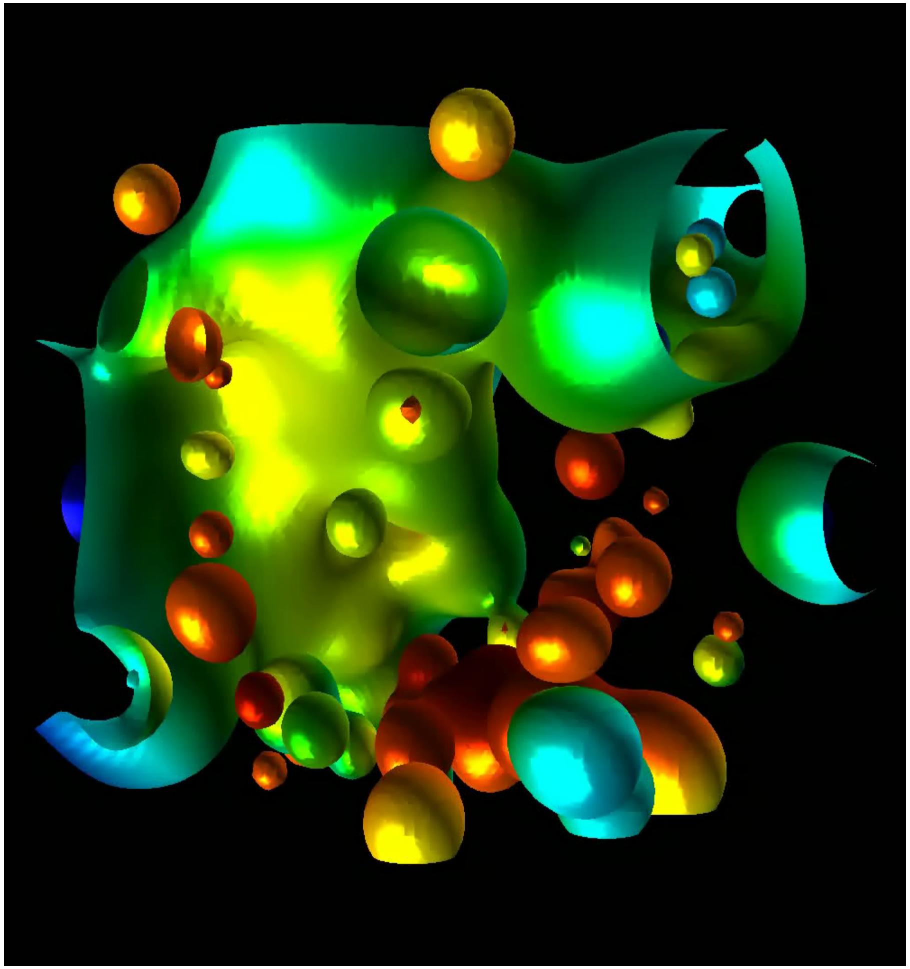
```
struct LightingParameters
{
    LightingParameters(void)
    {
         lightDir[0] = -0.6;
         lightDir[1] = 0;
         lightDir[2] = -0.8;
         Ka = 0.3;
         Kd = 0.7;
         Ks = 2.3;
         alpha = 2.5;
    };


    double 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.
};

LightingParameters lp;
```

Project #1F (8%), Due Feb 19th

- Goal: add shading, movie
- Extend your project1E code
- Important:
- add #define NORMALS



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 (with #define NORMALS) until you make these changes

→reader1e.cxx will initialize normals at each vertex

More comments (1/3)

- This project in a nutshell:
  - Add method called “CalculateShading”.
    - My version of CalculateShading is about ten lines of code.
  - Call CalculateShading for each vertex
  - This is a new field, which you will LERP.
  - Modify RGB calculation to use shading.

More comments (2/3)

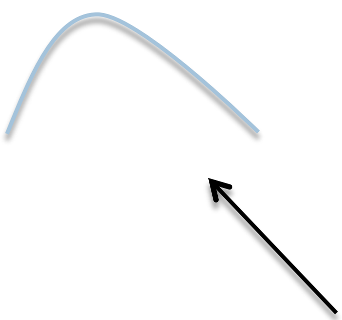
- New: more data to help debug
  - I will make the shading value for each pixel available.
  - I will also make it available for ambient, diffuse, specular.
- Don't forget to do two-sided lighting (for diffuse, not specular)

More comments (3/3)

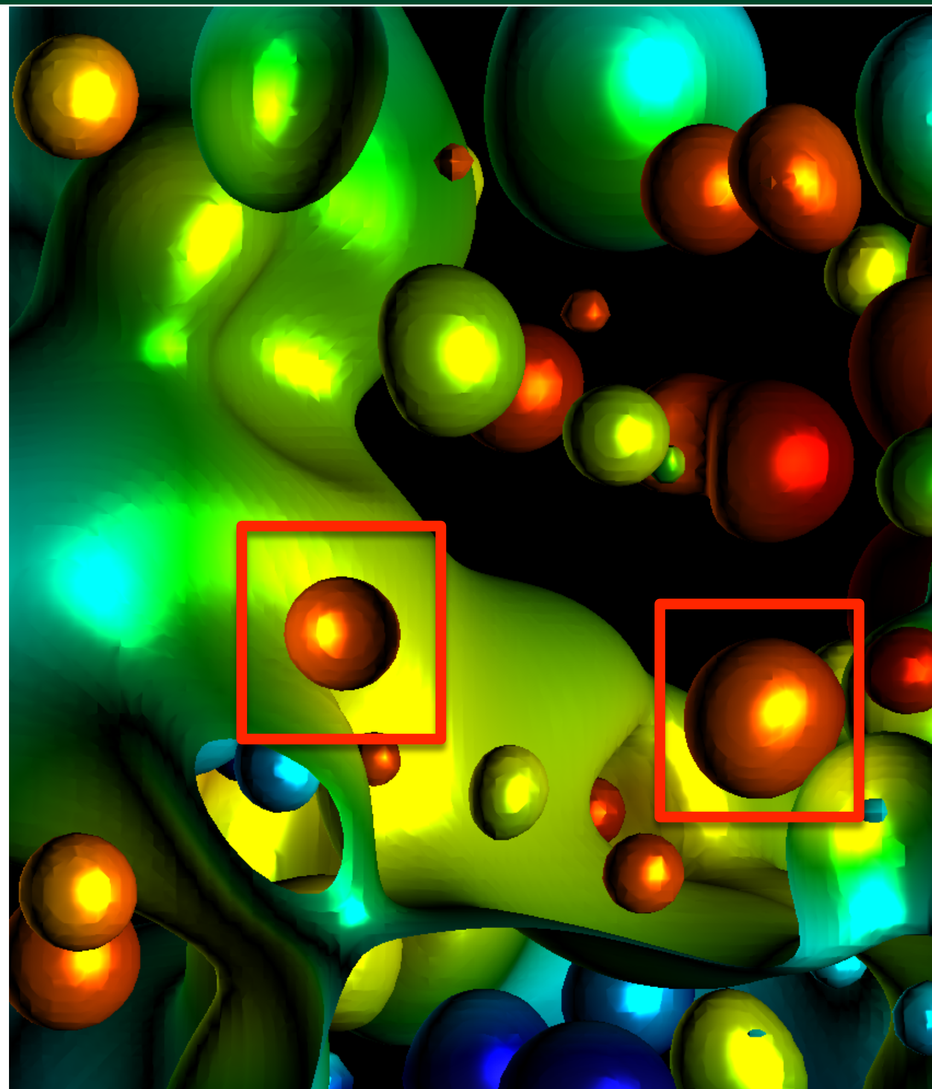
- I haven't said anything about movie encoders

Where Hank spent his debugging time…

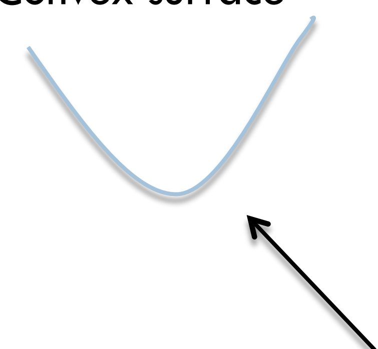
Concave surface



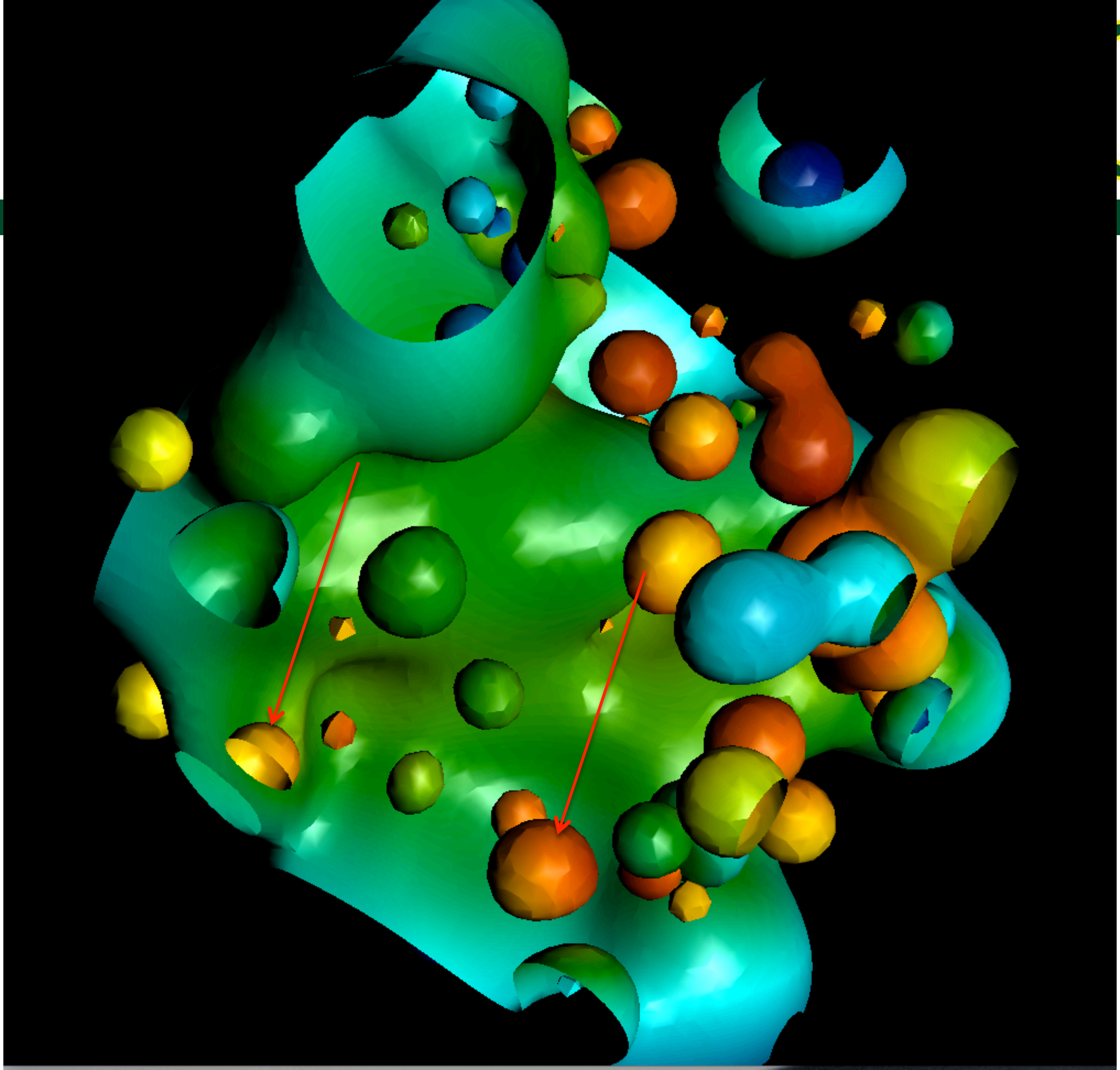
Lighting direction



Convex surface



Lighting direction



Project #1F (8%), Due Feb 19th

- Goal: add shading, movie