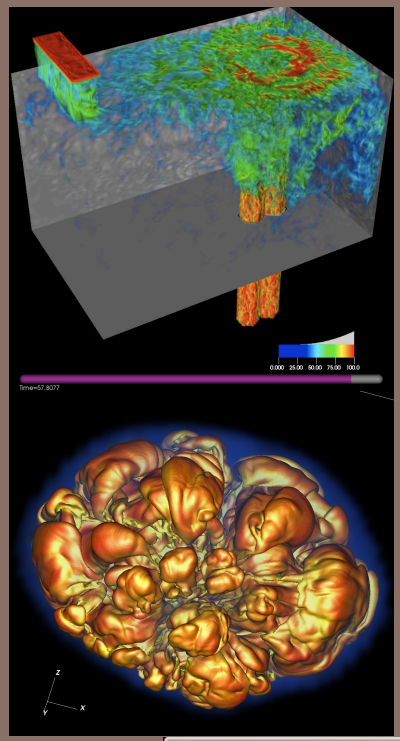
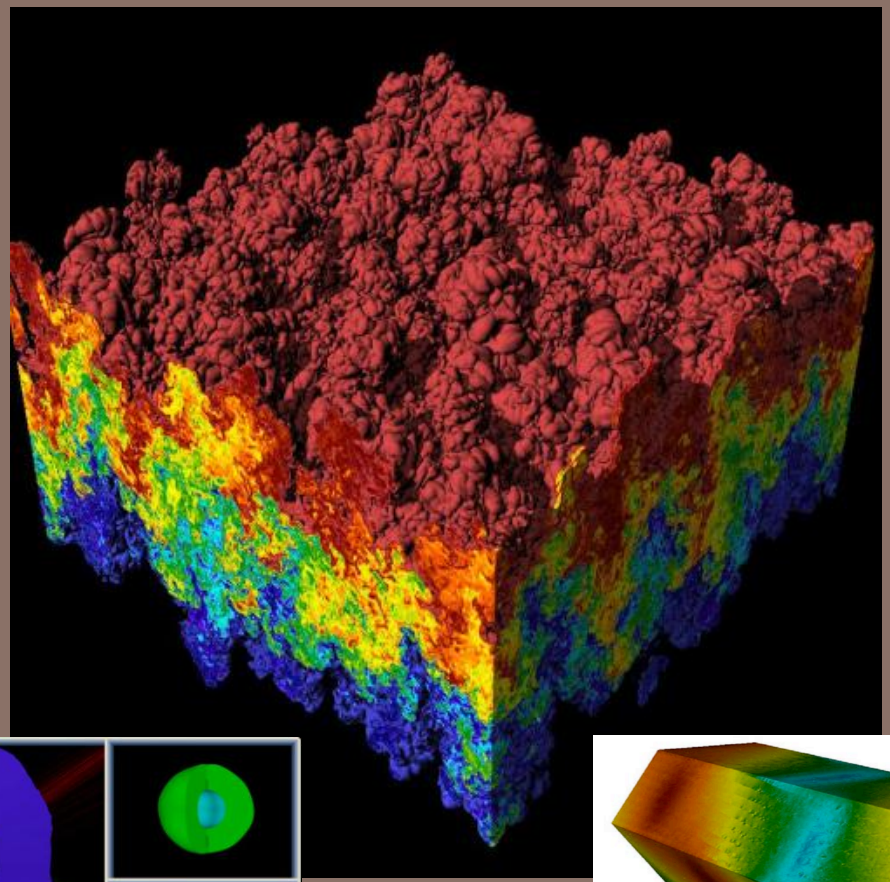
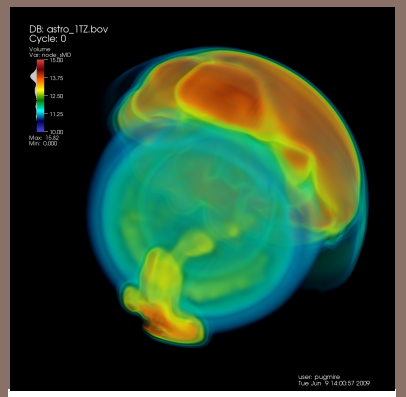
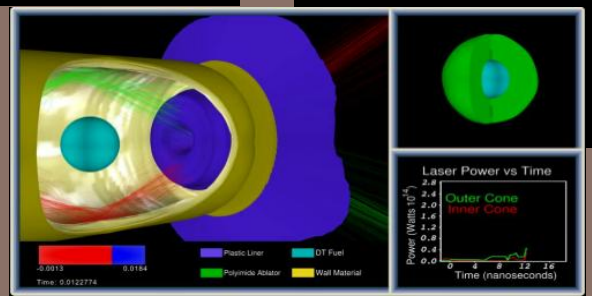
CIS 441/541: Introduction to Computer Graphics

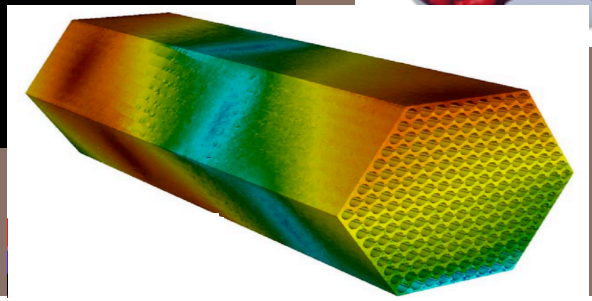
Lecture ~~8, 9~~: 10 Math Basics, Lighting Introduction & Phong Lighting











Oct. 17th, 2016

Hank Childs, University of Oregon

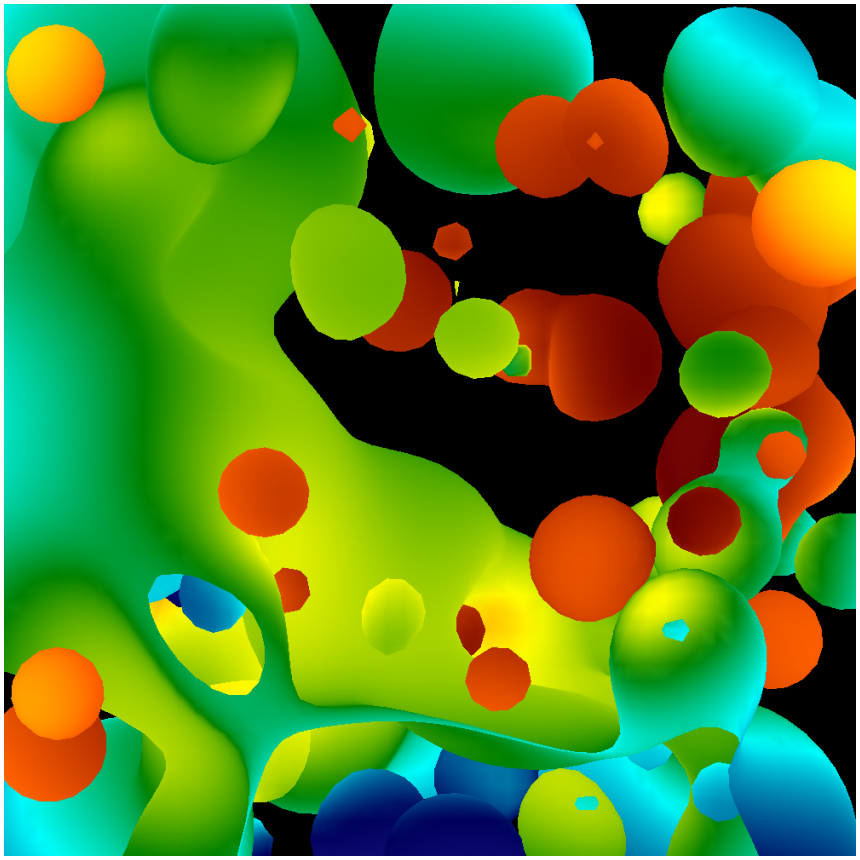
Announcements

Class Cancellation

- Class cancelled on Monday, October 24th
- Will do YouTube lecture (if necessary) to stay on track

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

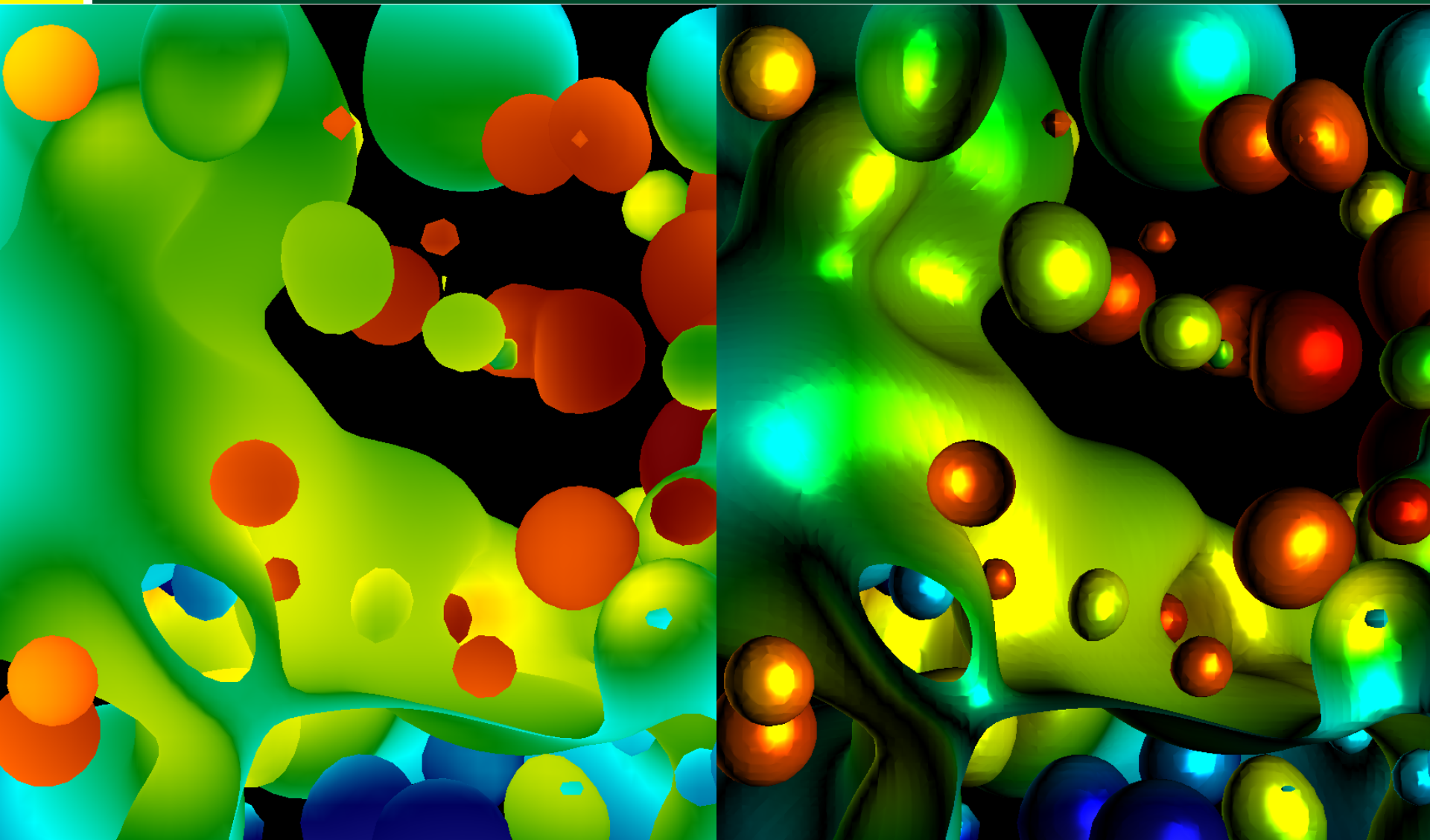
- Goal: interpolation of color and zbuffer
- Extend your project1C code
- File proj1d_geometry.vtk available on web (1.4MB)
- File “reader1d.cxx” has code to read triangles from file.
- No Cmake, project1d.cxx



QuackCon Extension

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

Our goal with 1E: add shading



Shading

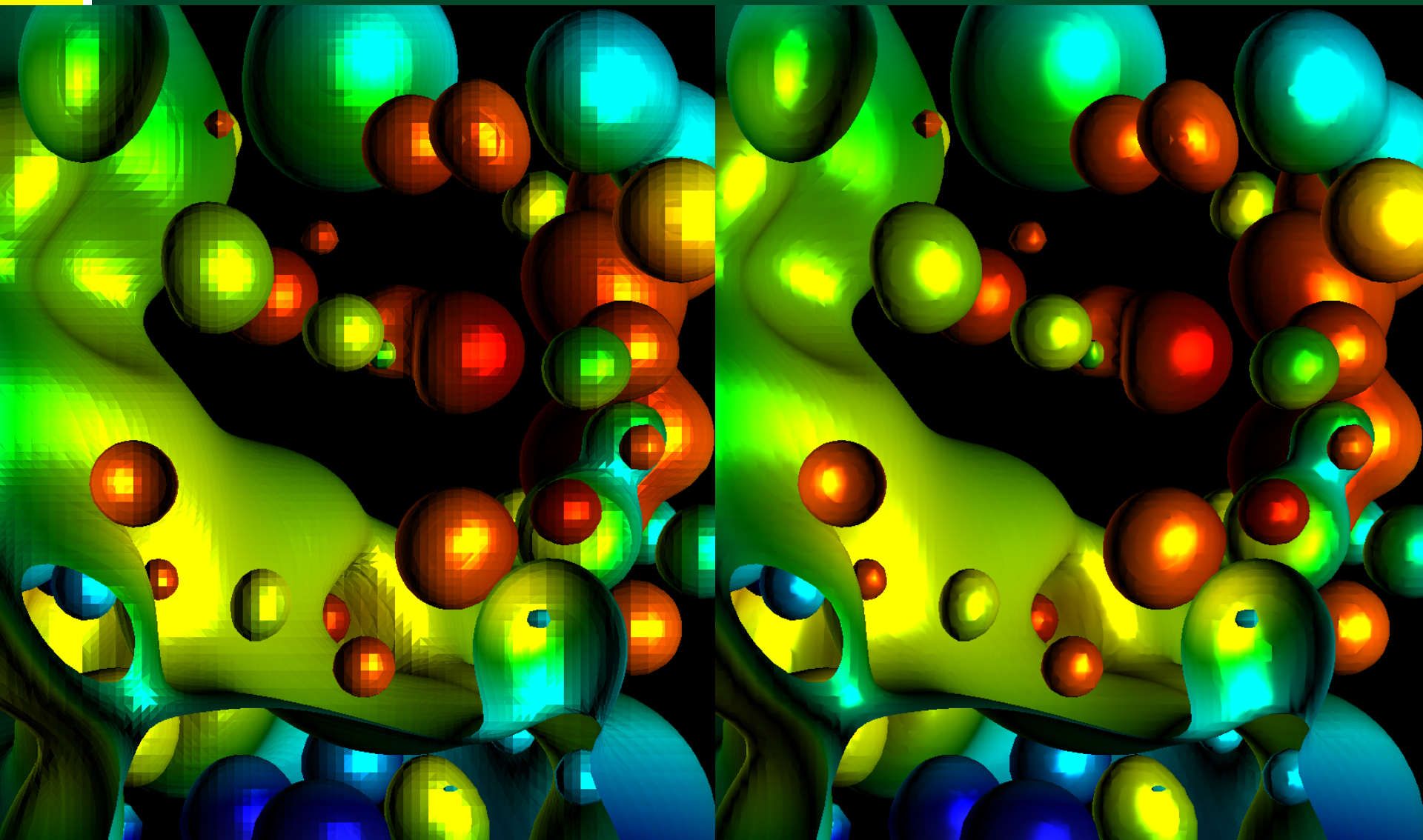
- Shading != Shadows
  - Shading:
    - brighter where light hits square
    - darker where light is tangent
  - Shadow: one object obscures light from getting to another object
- Shading & Normals
  - Normals – orientation of geometry with respect to light source
  - Key ingredient in recipe for shading
    - (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

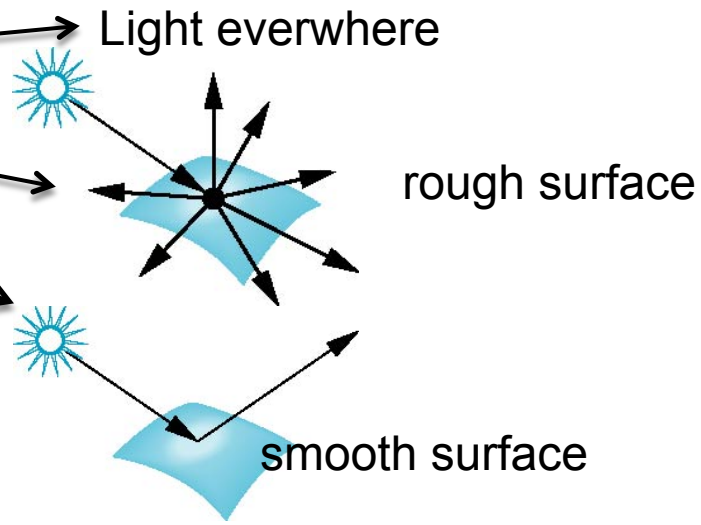
- Constant over a triangle ←→ flat shading
  - Take (C-A)x(B-A) as normal over whole triangle
- Varying over a triangle ←→ smooth shading
  - Calculate normal at vertex, then use linear interpolation
    - How do you calculate normal at a vertex?
    - How do you linearly interpolate normals?

Outline

- Math Basics
- Lighting Basics
- The Phong Model

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.

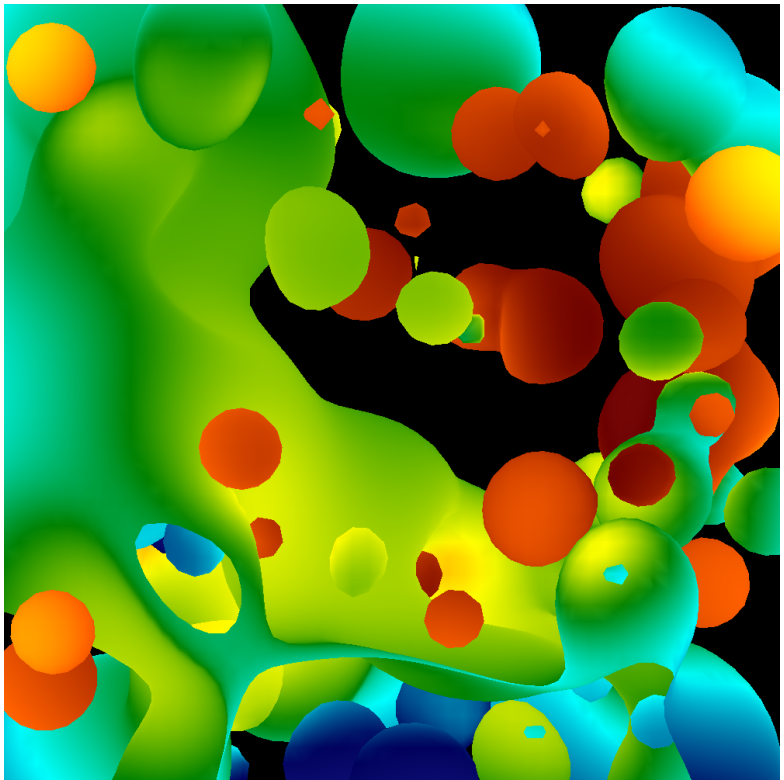
How to handle shading values greater than 1?

- 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

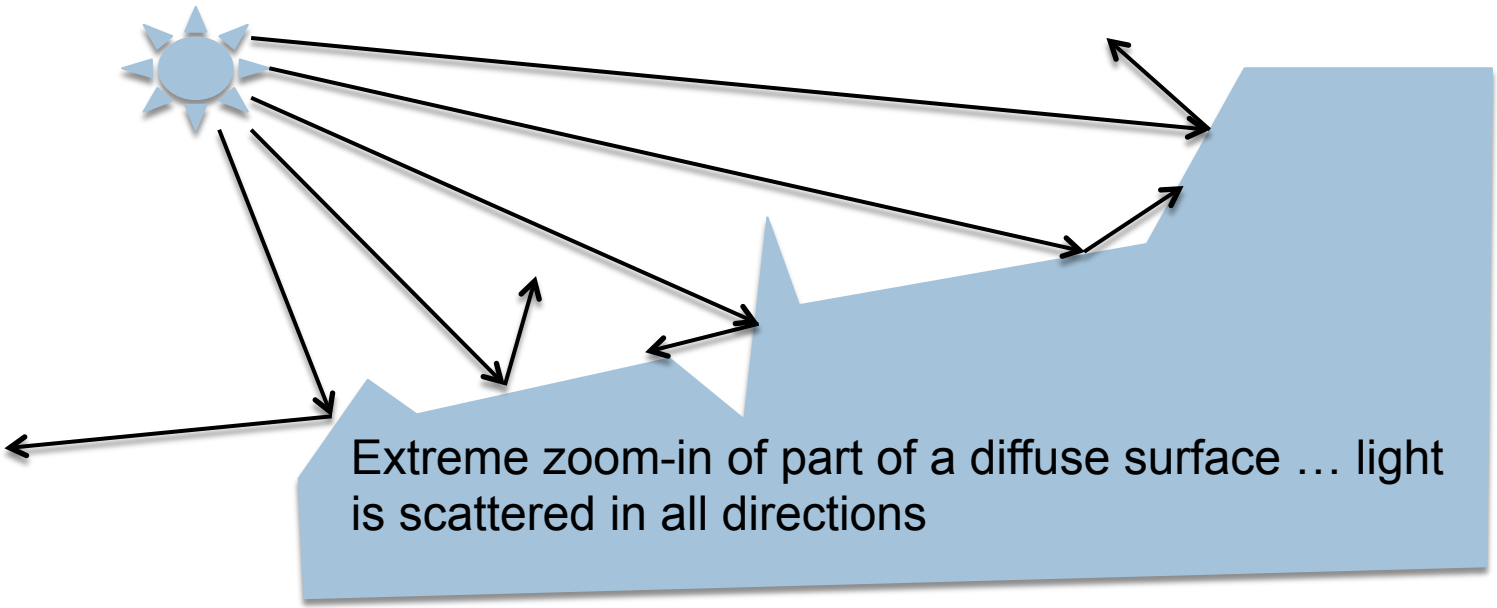
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Surface lit with ambient lighting only



Lambertian Surface

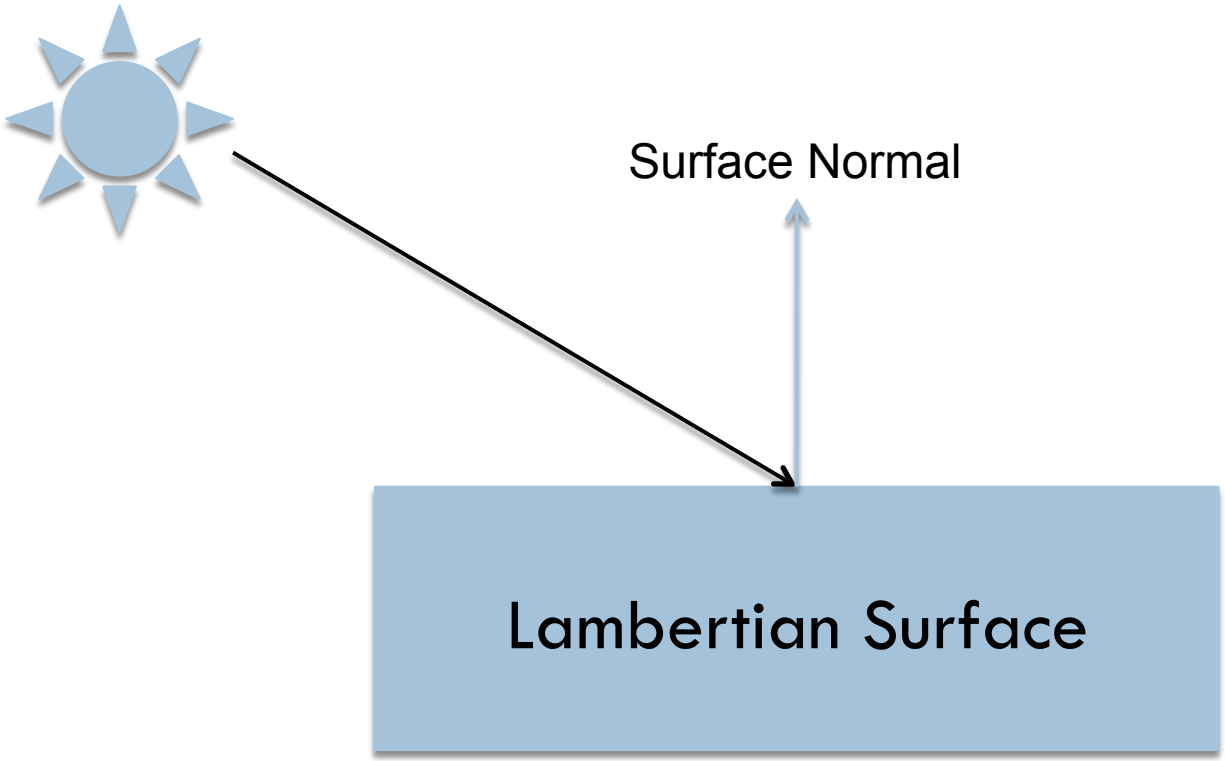
- Perfectly diffuse reflector
- Light scattered <u>equally</u> in all directions



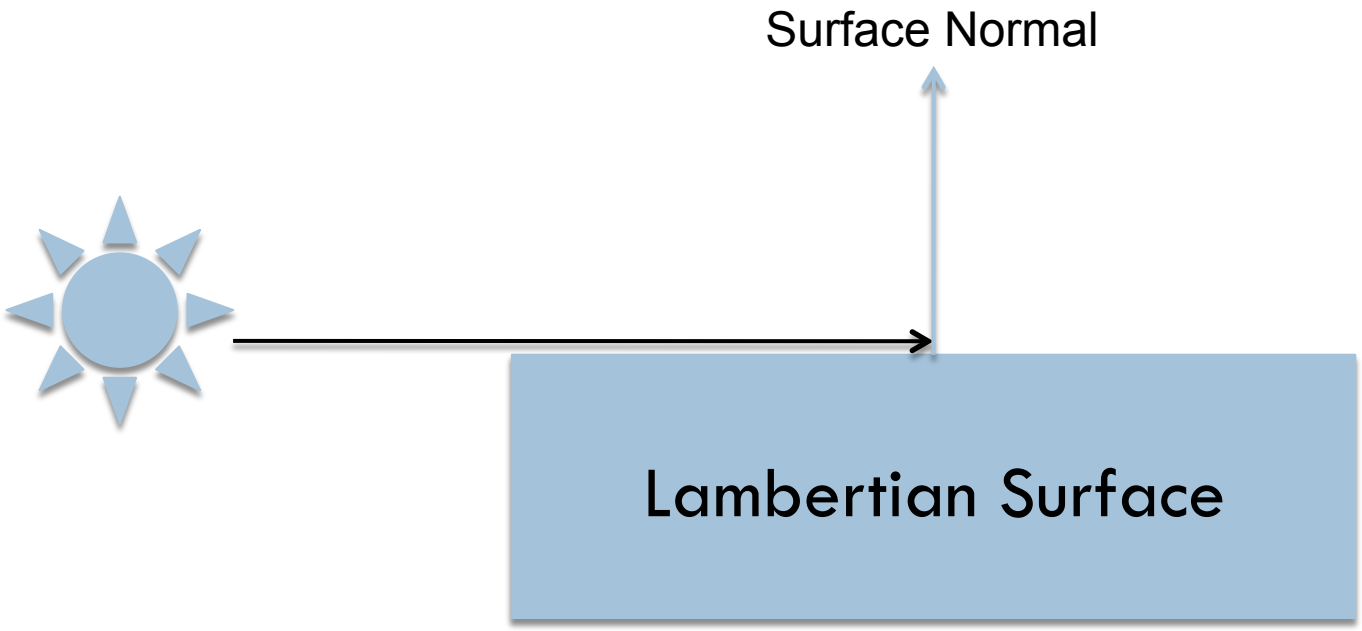
(this image shows 5 of the directions)

Slide inspired by Ed Angel Computer Graphics Book

Diffuse Lighting

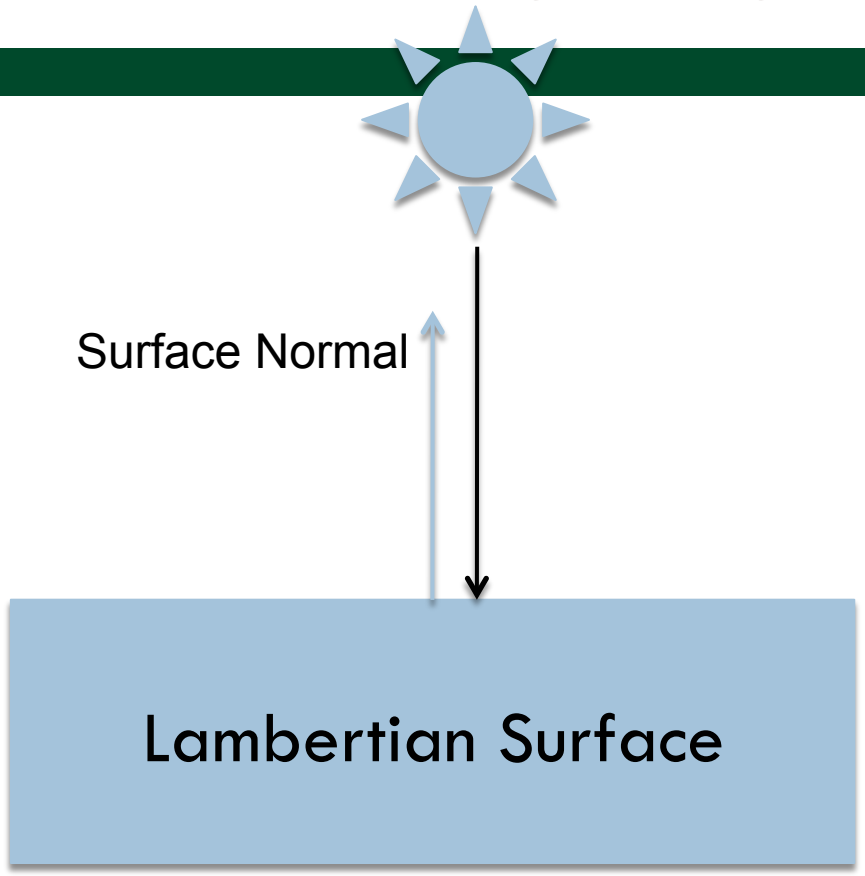


Diffuse Lighting



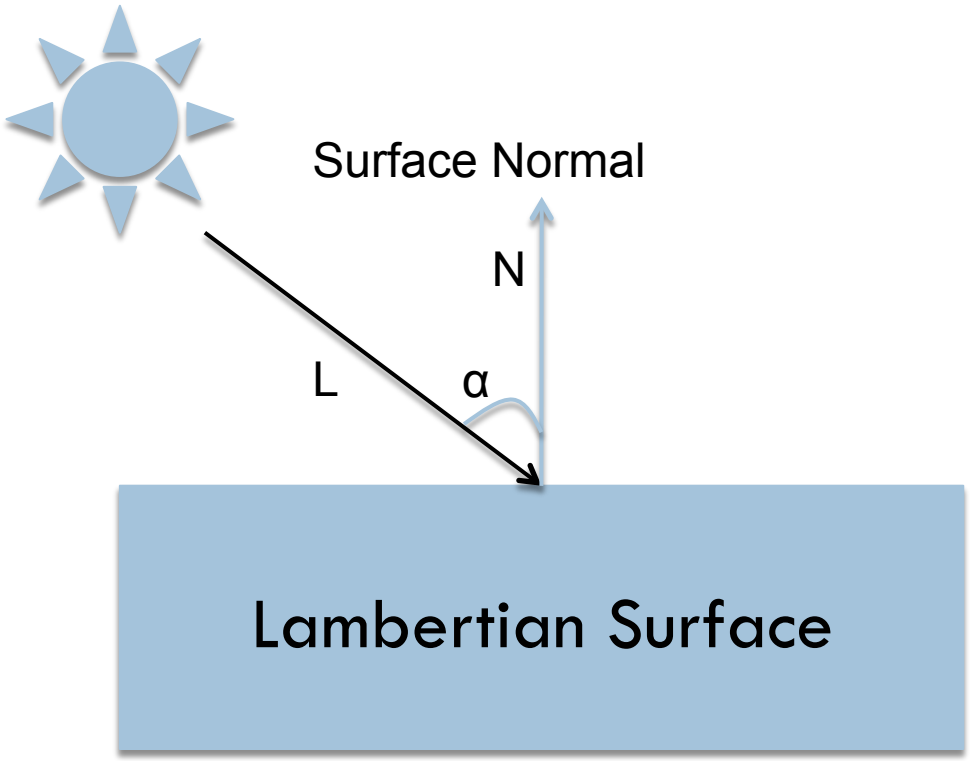
No light reflects off the (top) surface
(Light direction and surface normal are perpendicular)

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?

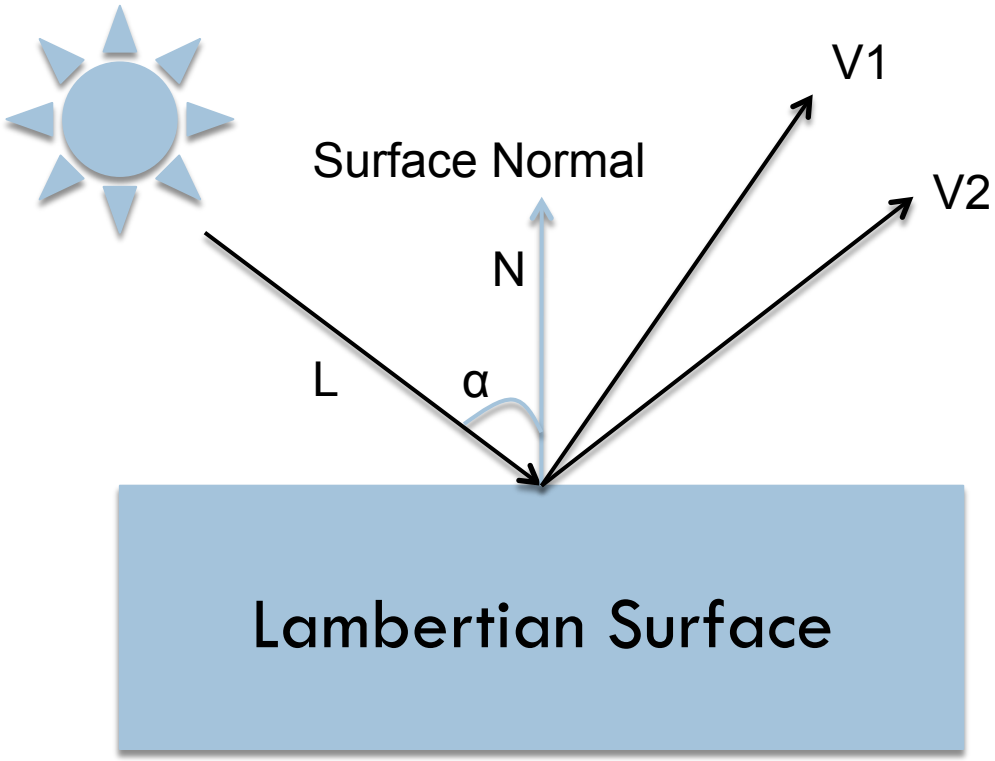
A: cos(α)

And note that:

cos(0) = 1

cos(90) = 0

Diffuse Lighting



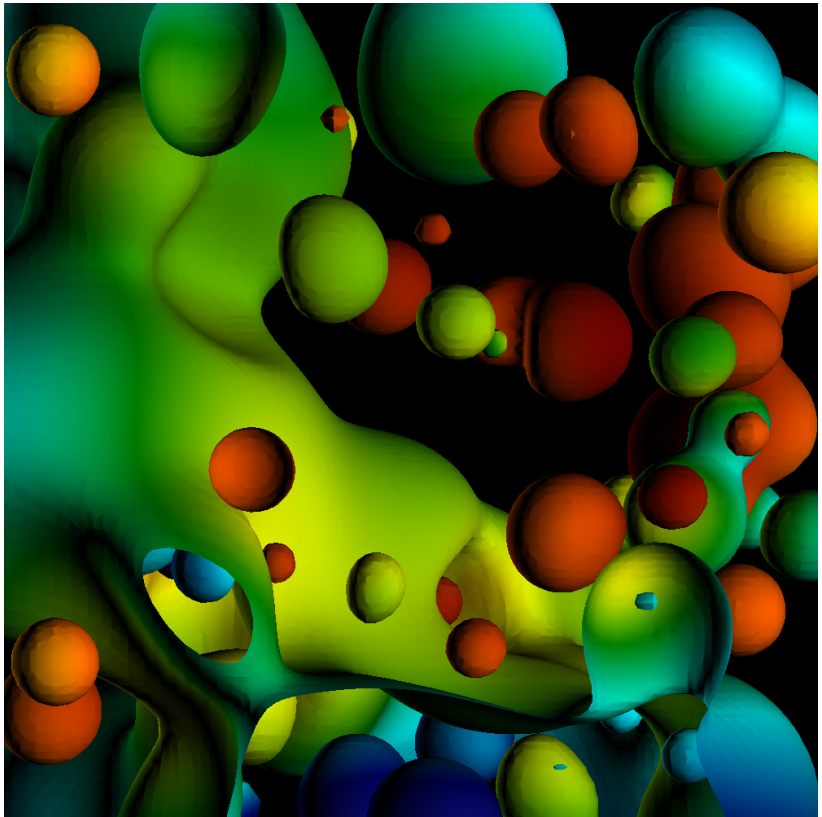
How much light makes it to viewer V1? Viewer V2?

A: cos(α) for both

Lambertian surfaces reflect light equally in all directions

Diffuse Lighting

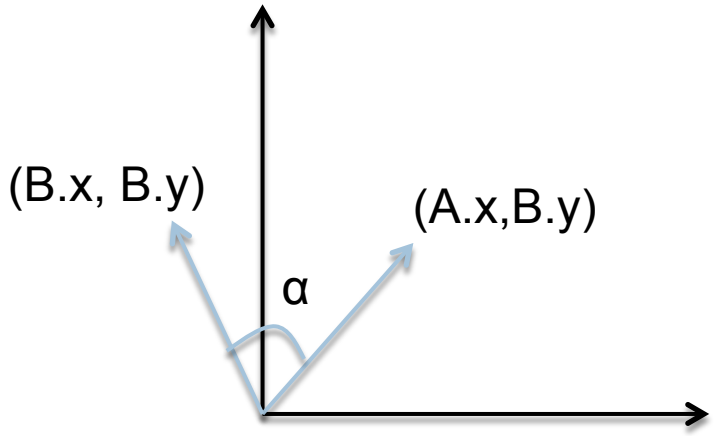
- Diffuse light
  - Light distributed evenly in all directions, but amount of light depends on orientation of triangles with respect to light source.
  - Different for each triangle



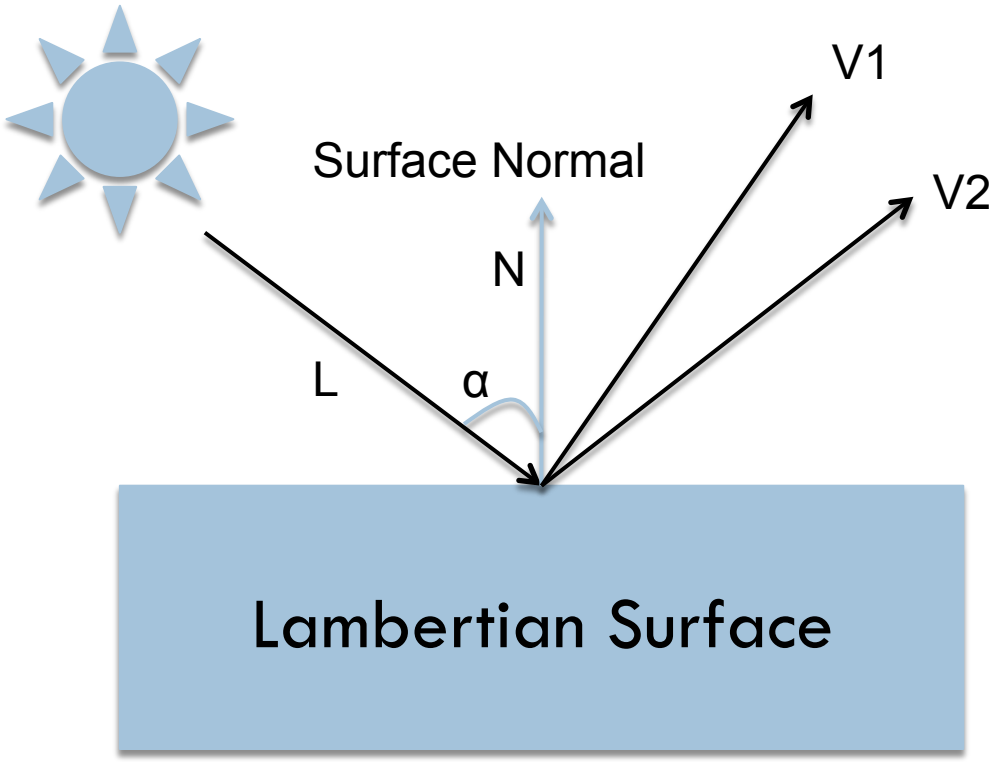
Surface lit with diffuse lighting only

What is a dot product?

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

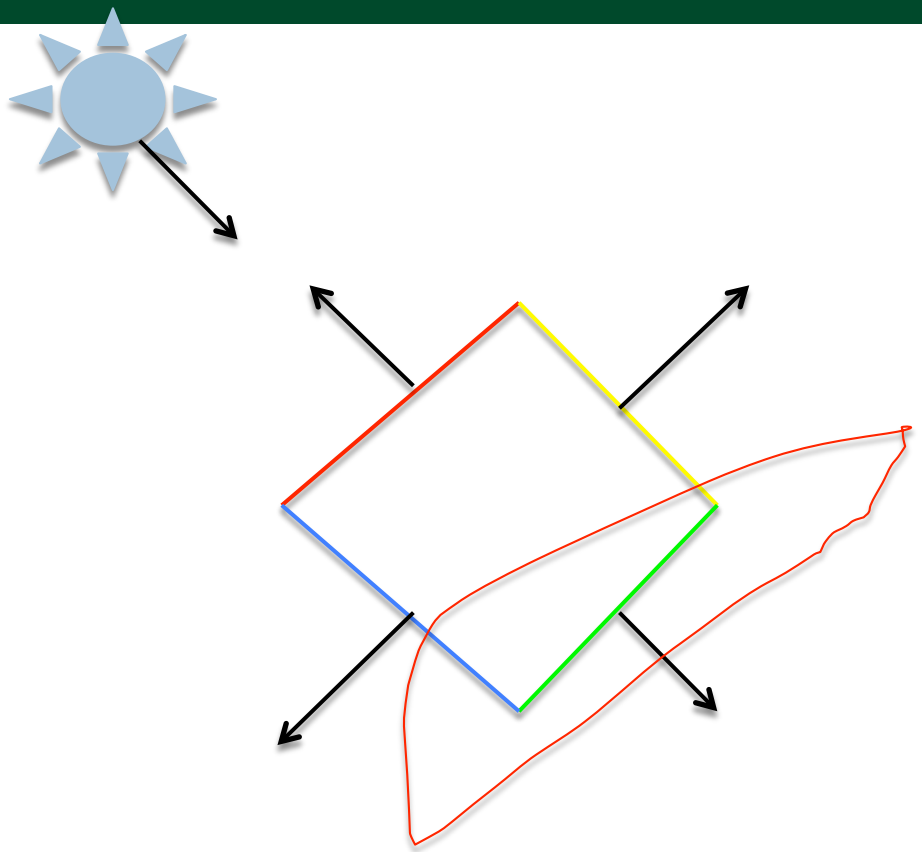


Diffuse Lighting

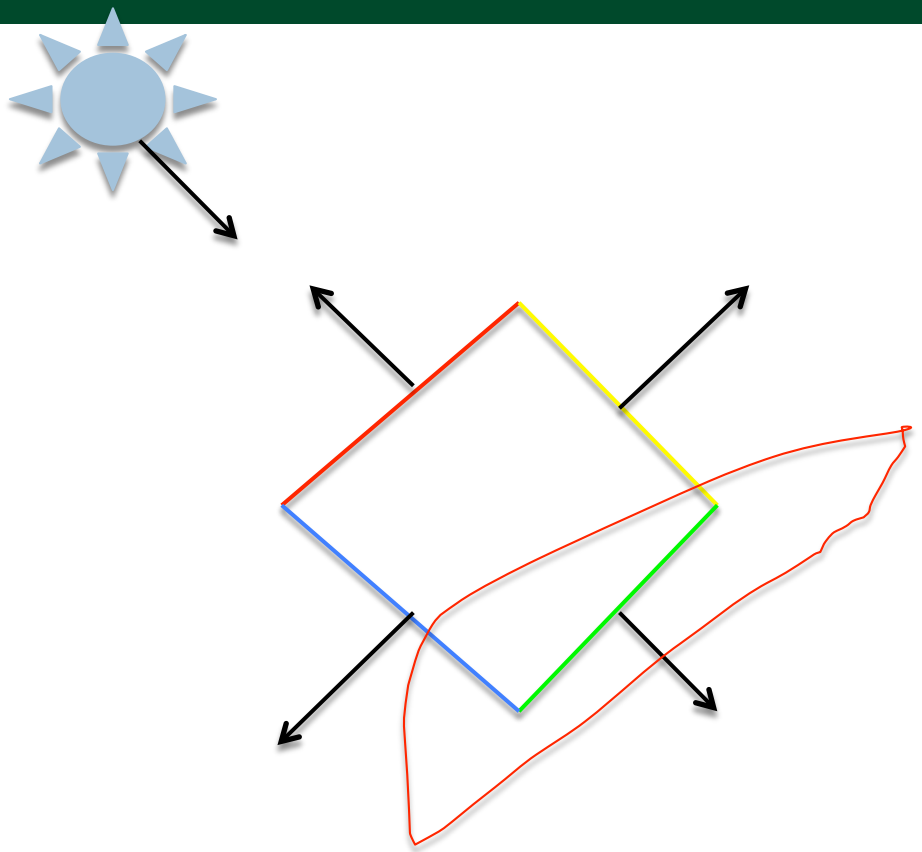


You can calculate the diffuse contribution by taking the dot product of L and N,
Since $L \cdot N = \cos(\alpha)$
(assuming L and N are normalized)

What about cases where L·N < 0?

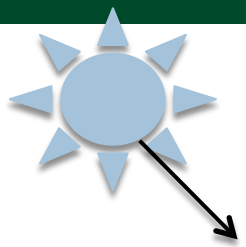


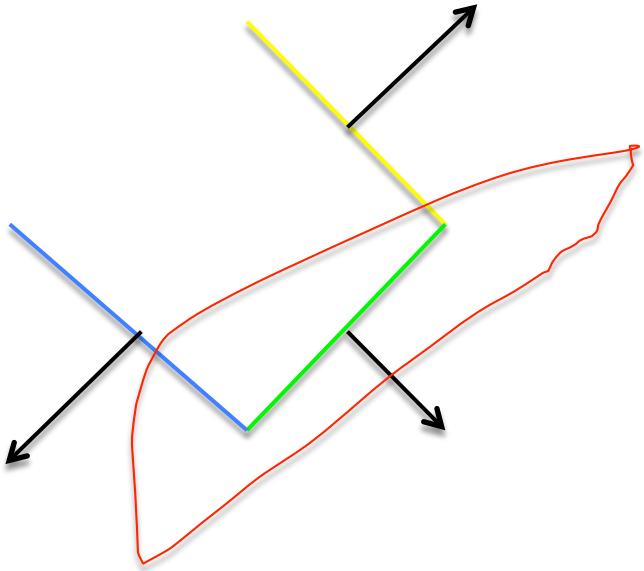
What about cases where L·N < 0?



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

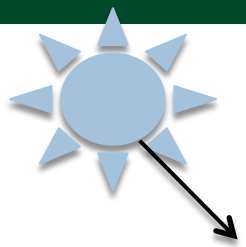
But wait…

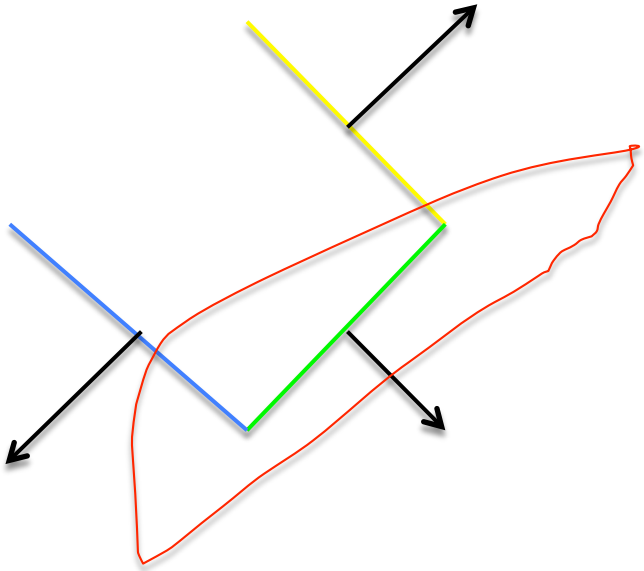




If you have an open surface, then there is a “back face”. The back face has the opposite normal.

But wait…

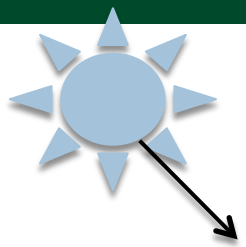


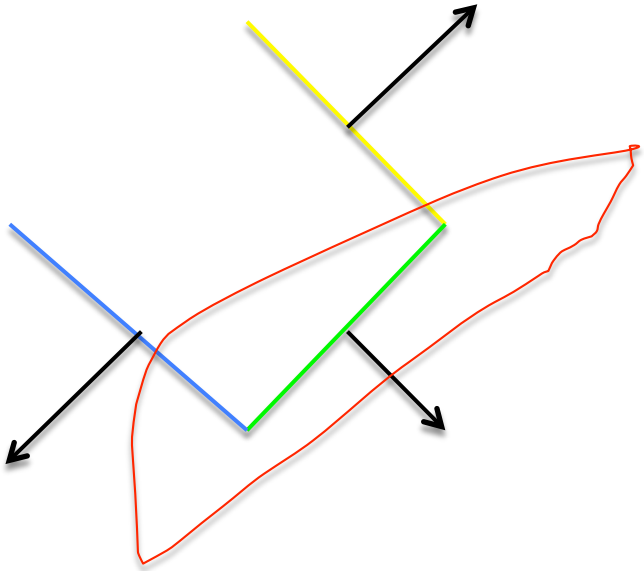


If you have an open surface, then there is a “back face”. The back face has the opposite normal.

How can we deal with this case?

But wait…



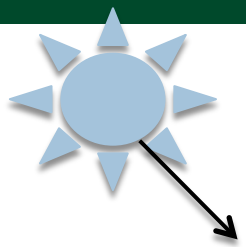


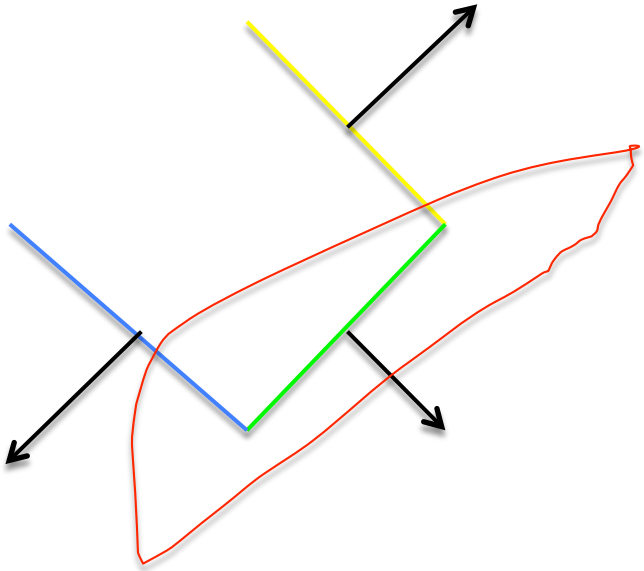
If you have an open surface, then there is a “back face”. The back face has the opposite normal.

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?

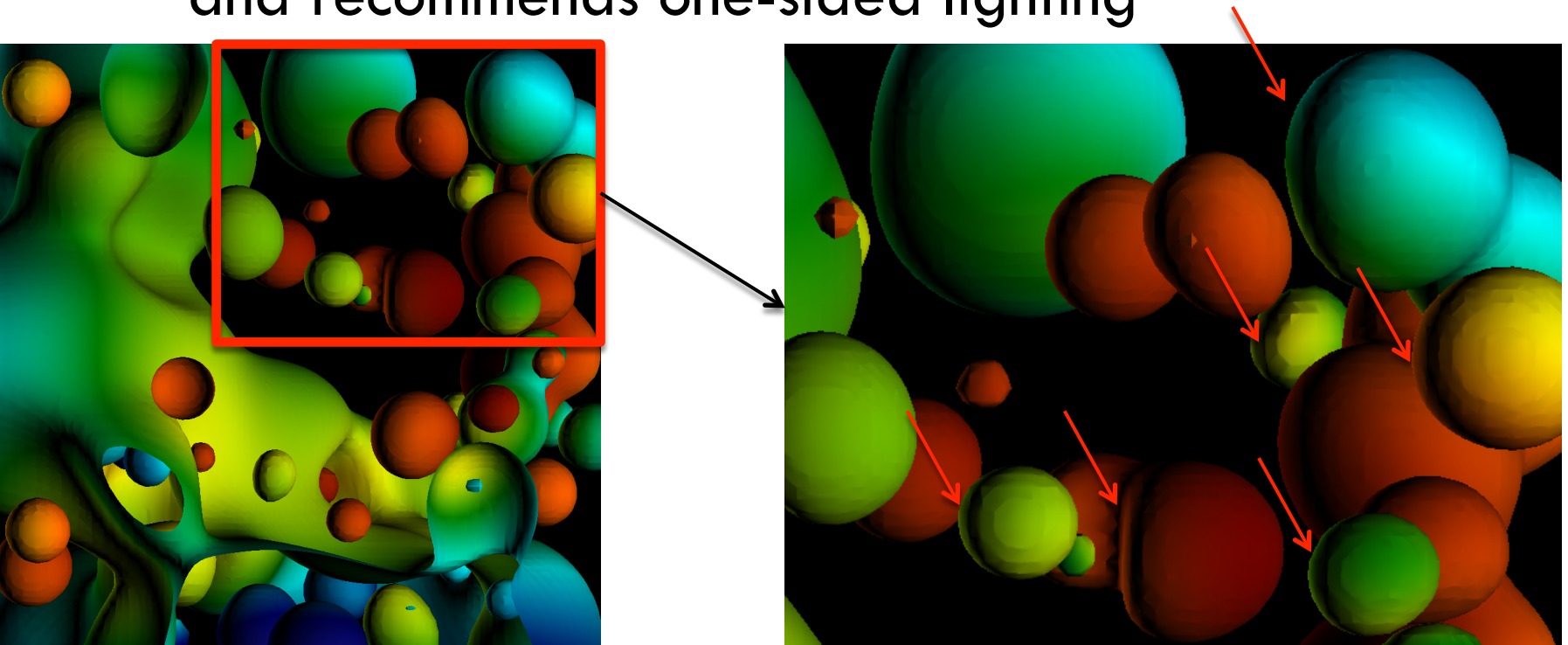
Idea #1: encode all triangles twice, with different normals
Idea #2: modify diffuse lighting model

Diffuse light = abs(L·N)

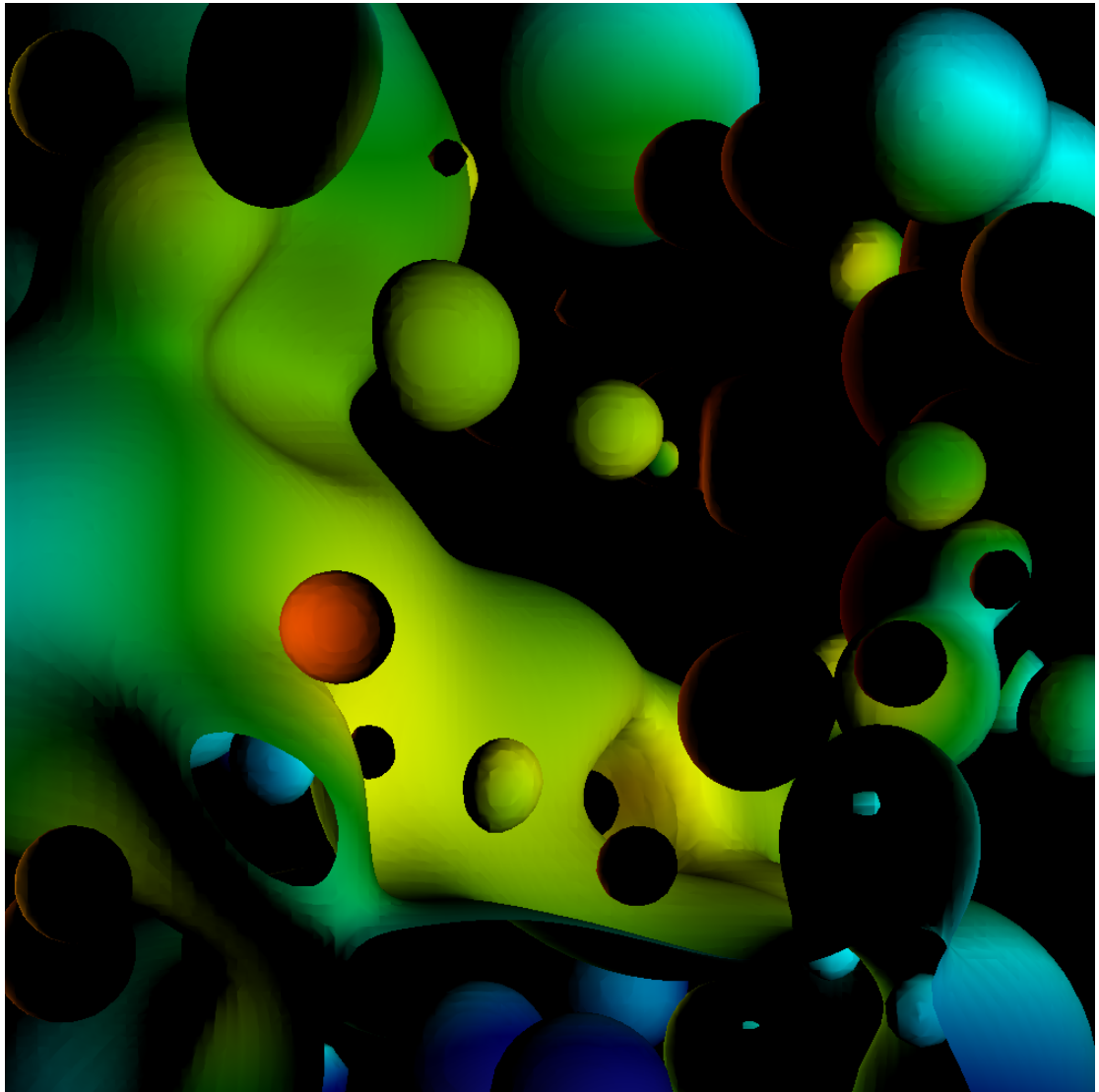
This is called two-sided lighting

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

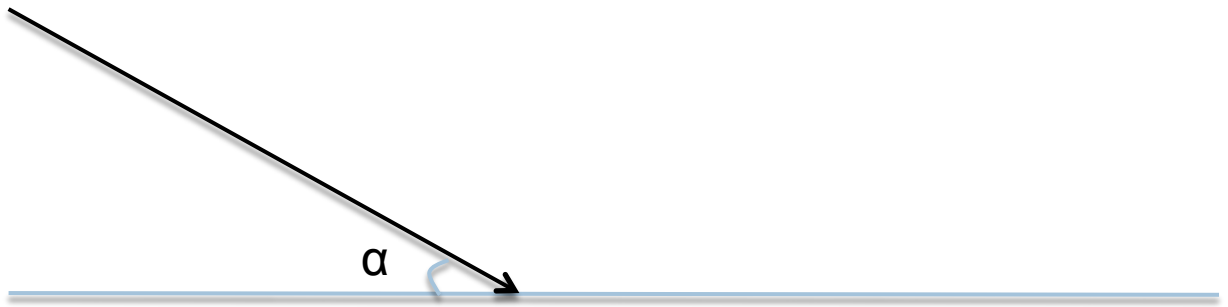


One-sided lighting with open surfaces is disappointing



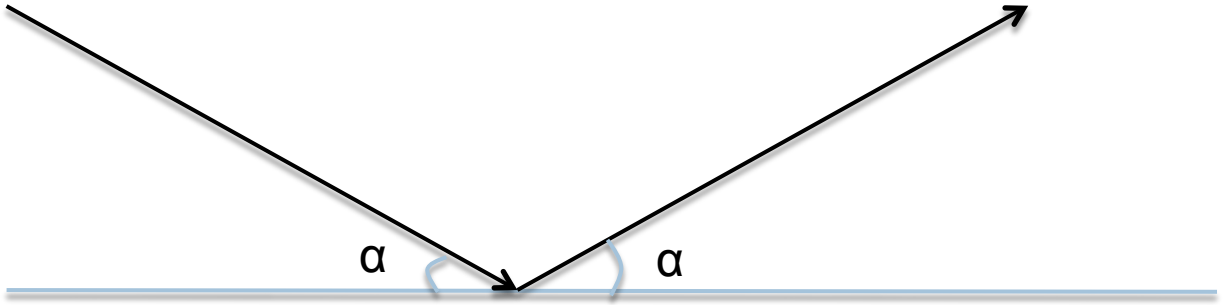
The most valuable thing I learned in Freshman Physics

- "angle in = angle out"

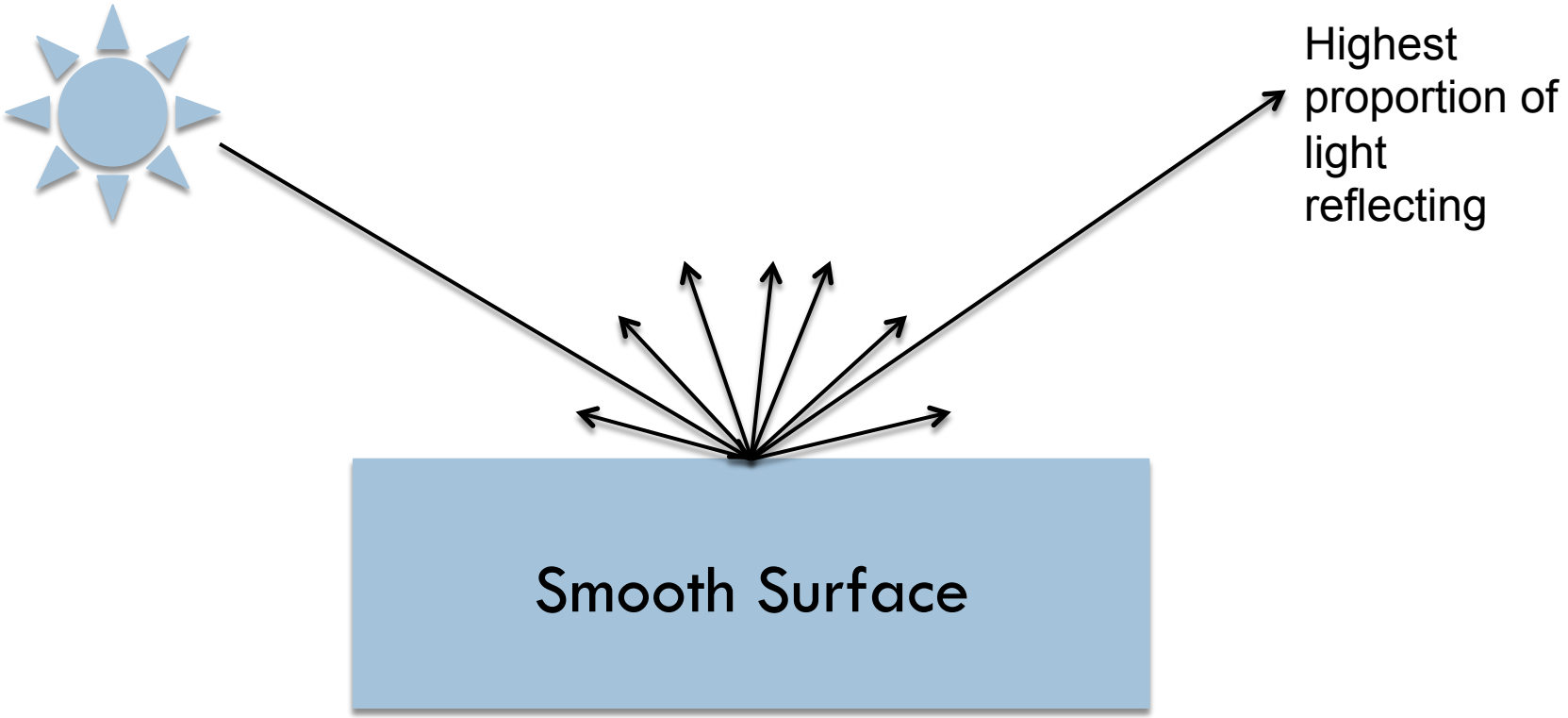


The most valuable thing I learned in Freshman Physics

- "angle in = angle out"

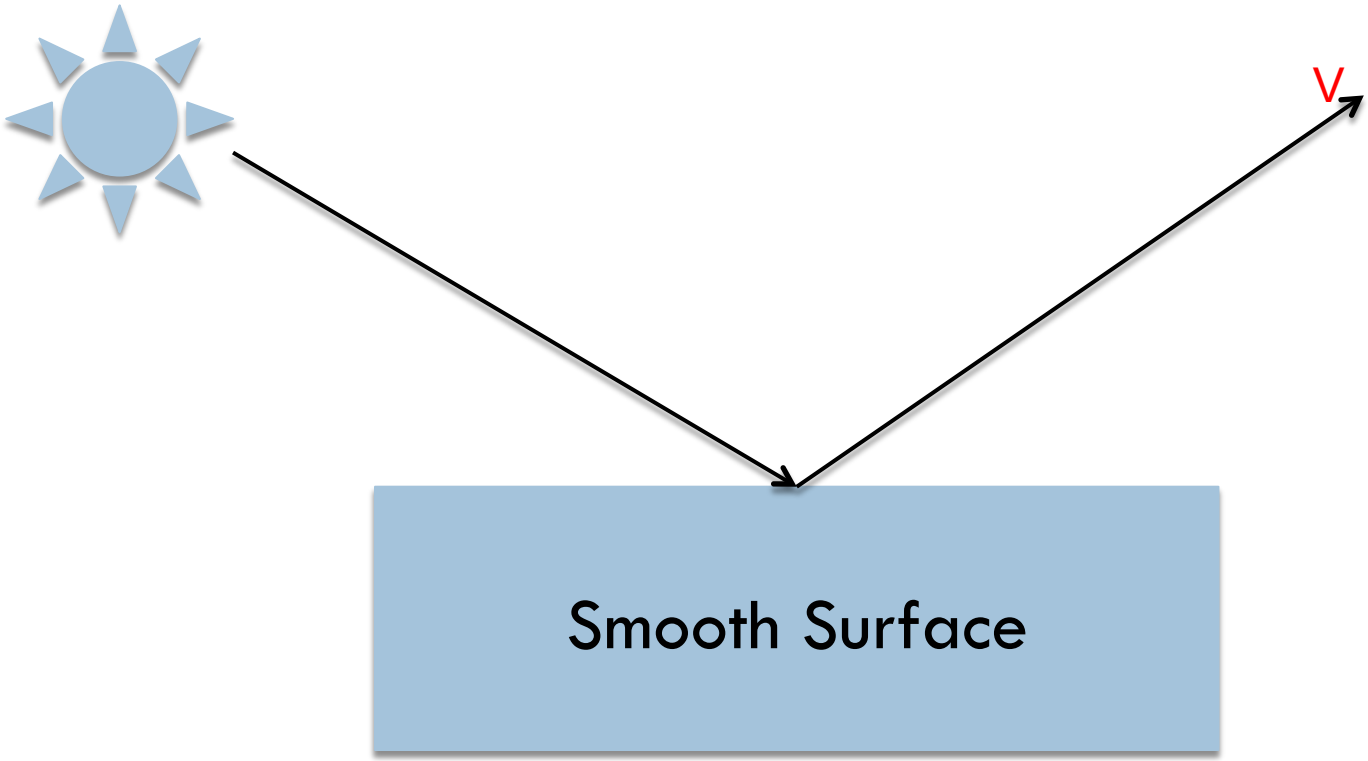


Specular Lighting



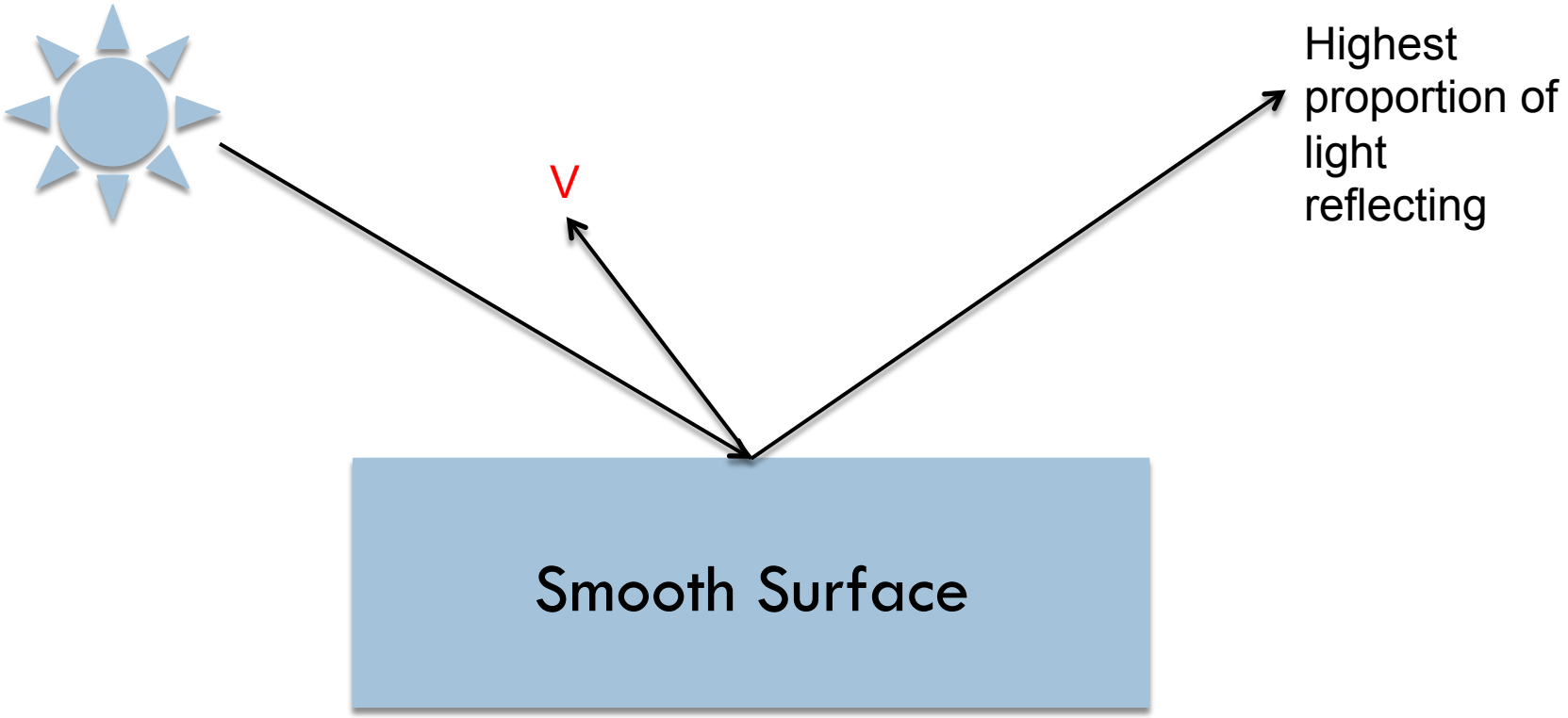
Light reflects in all directions.
But the surface is smooth, not Lambertian, so amount of reflected light varies.
So how much light??

How much light reflects with specular lighting?



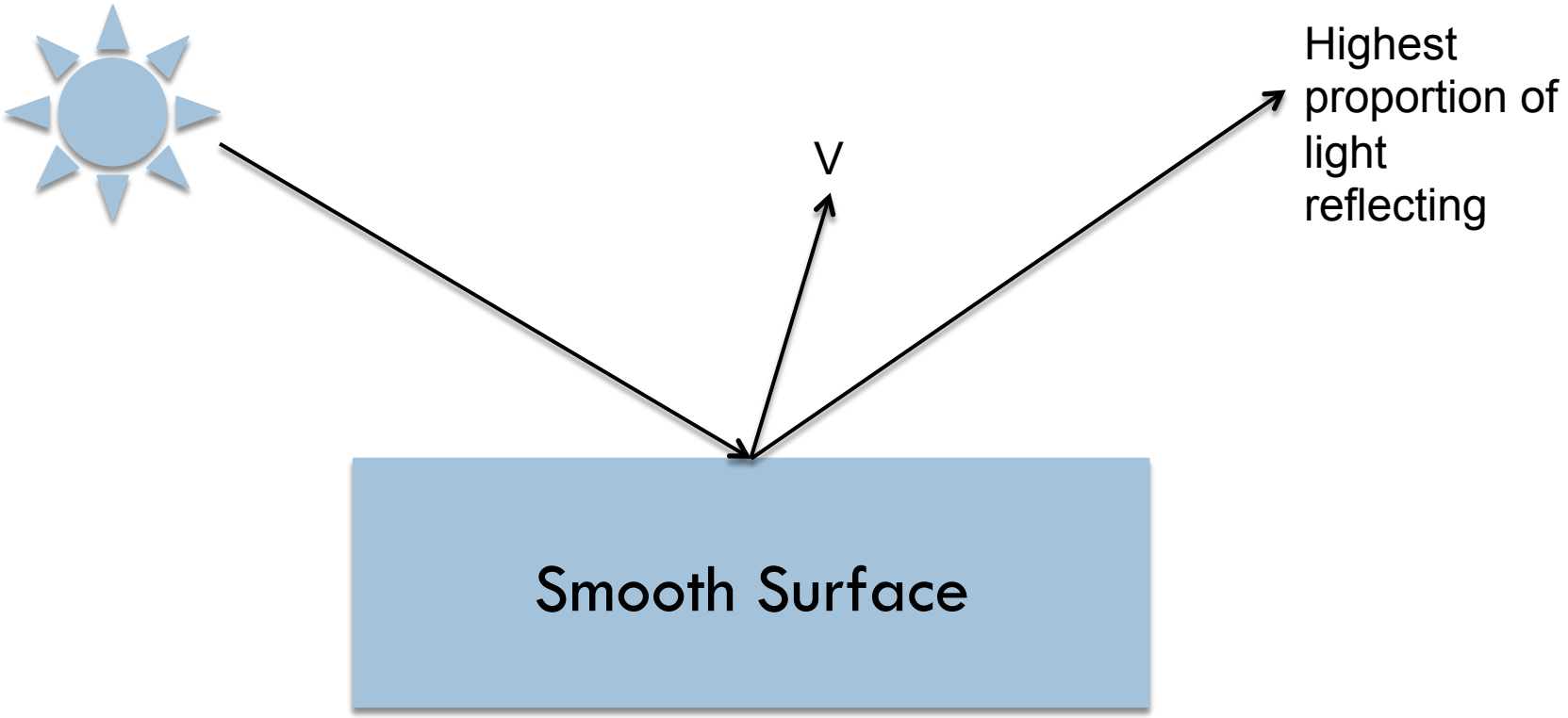
Consider V located along reflection ray.
Answer: most possible
Call this “1”

How much light reflects with specular lighting?



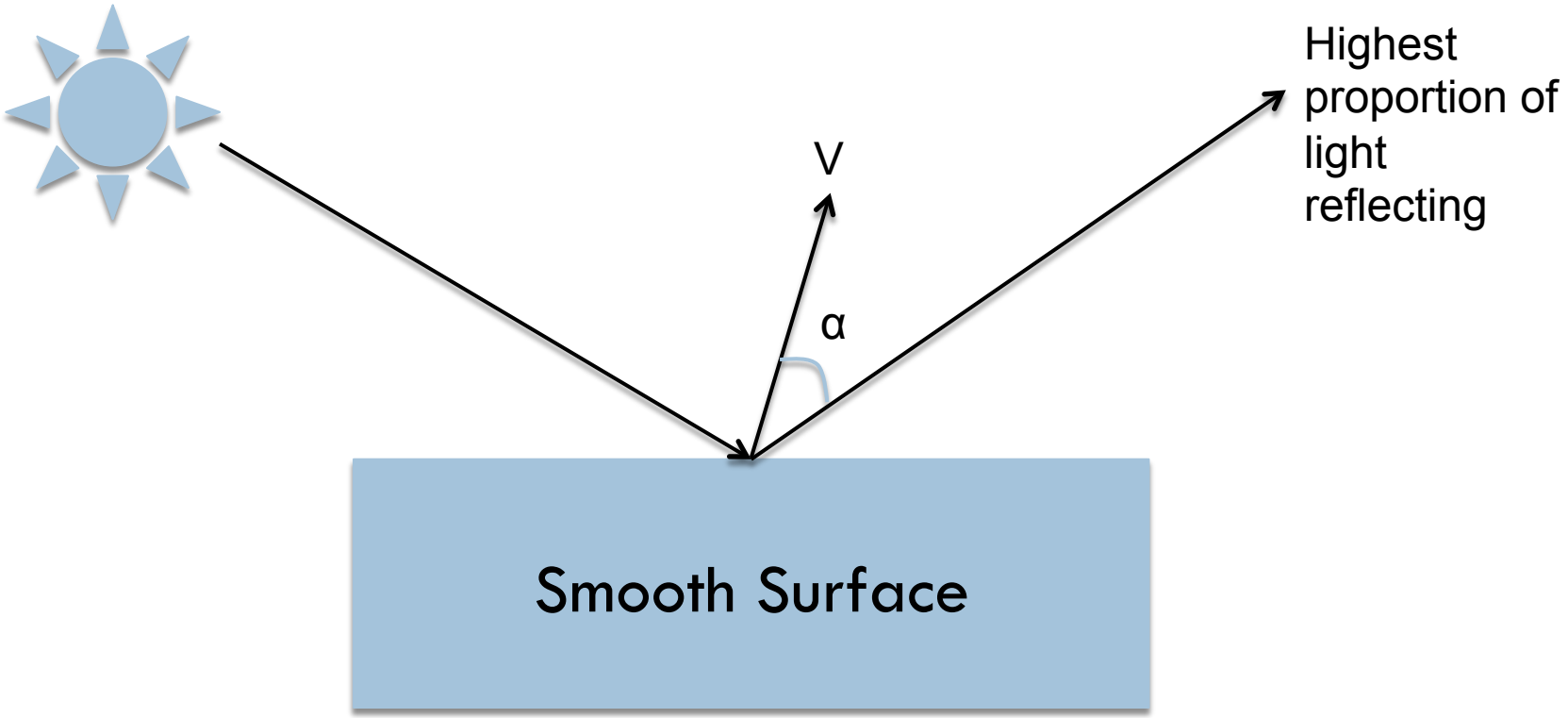
Consider V located along perpendicular ray.
Answer: none of it
Call this “0”

How much light reflects with specular lighting?



How much light gets to point V?

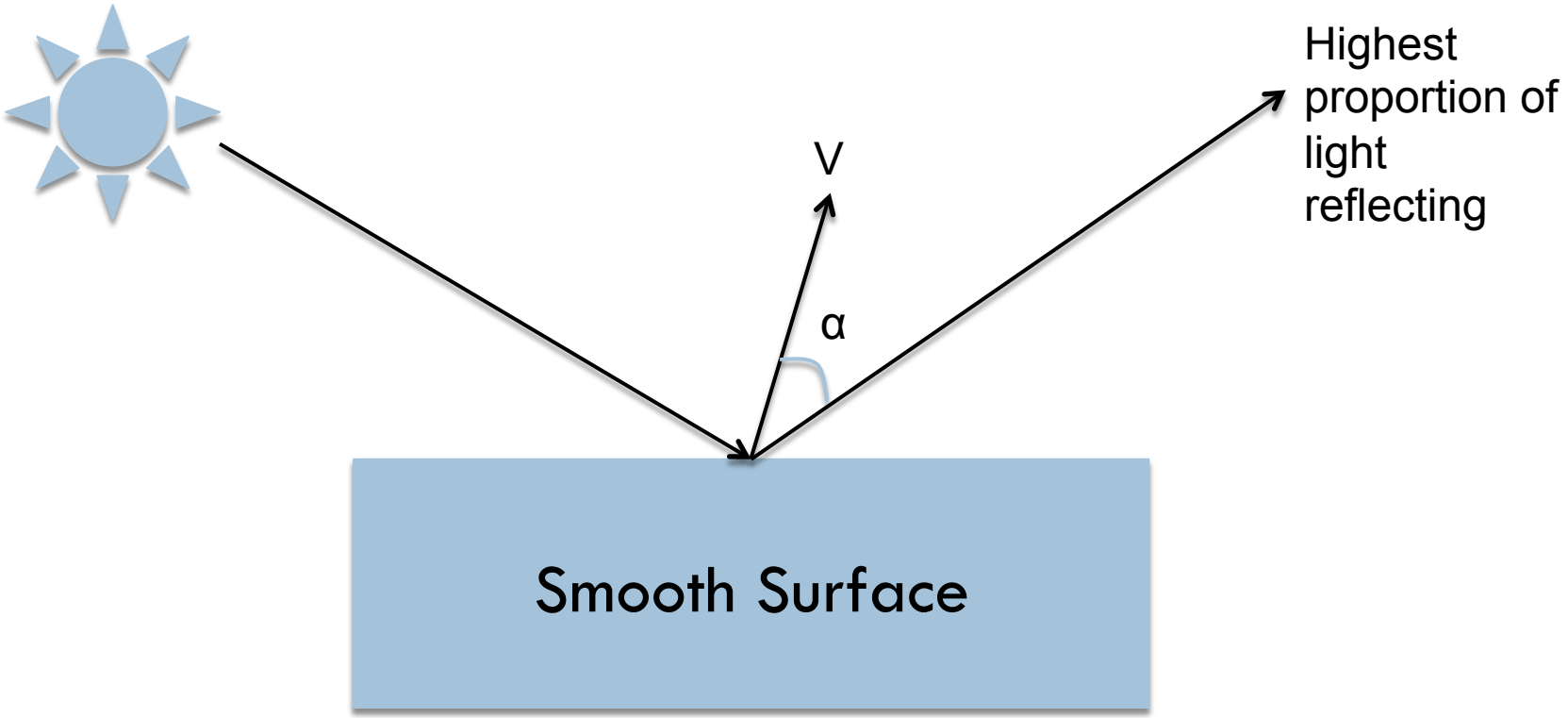
How much light reflects with specular lighting?



How much light gets to point V?

A: proportional to cos(α)

How much light reflects with specular lighting?

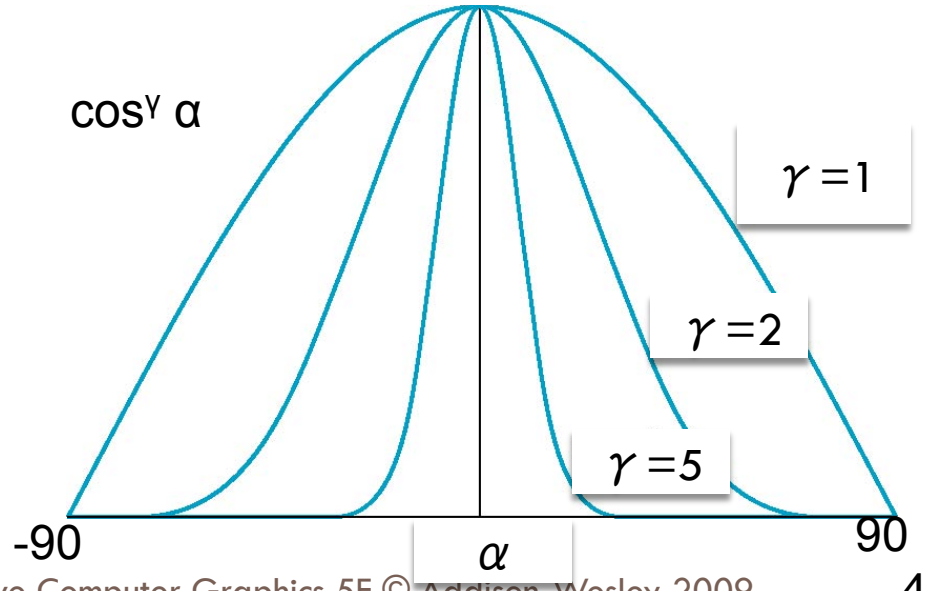


How much light gets to point V?

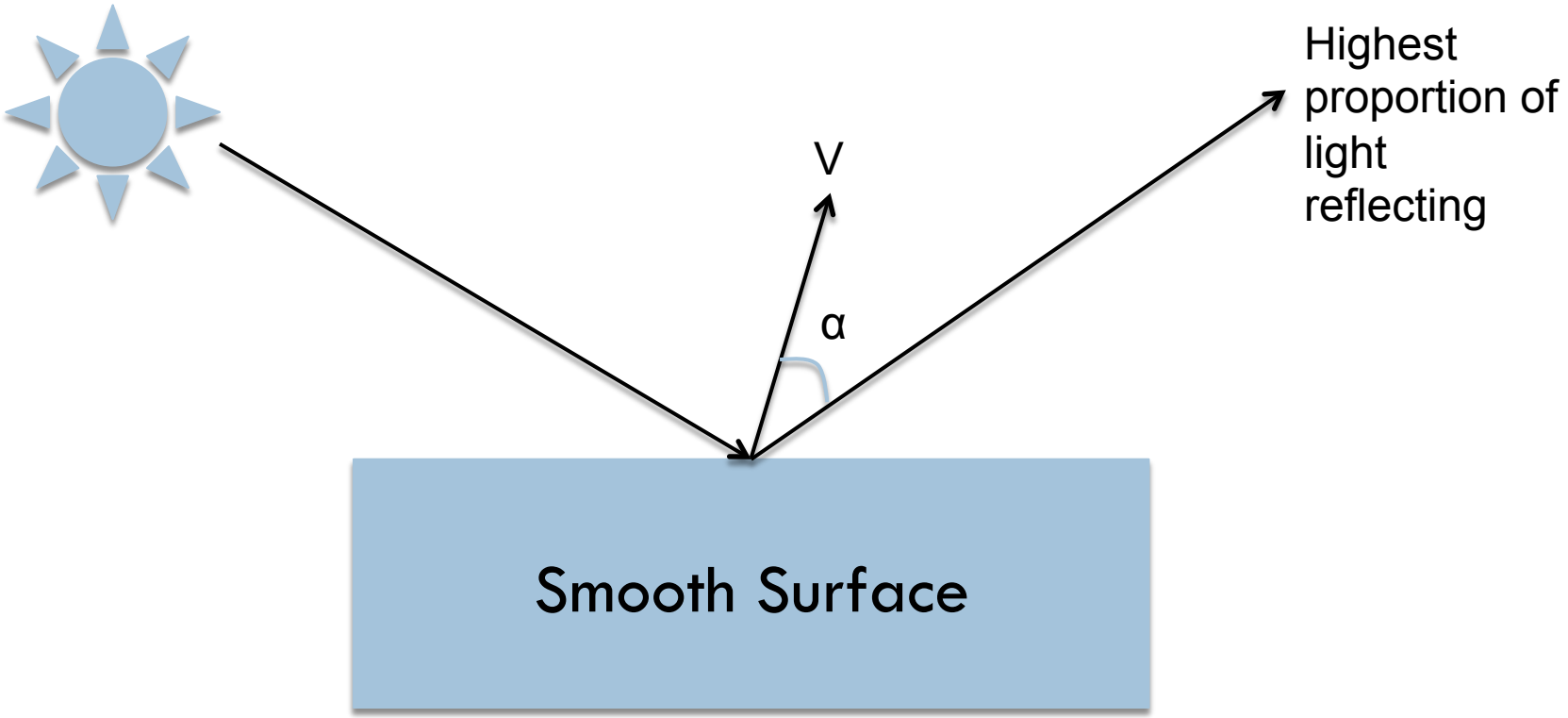
A: proportional to cos(α)
(Shininess strength) * cos(α) ^ (shininess coefficient)

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



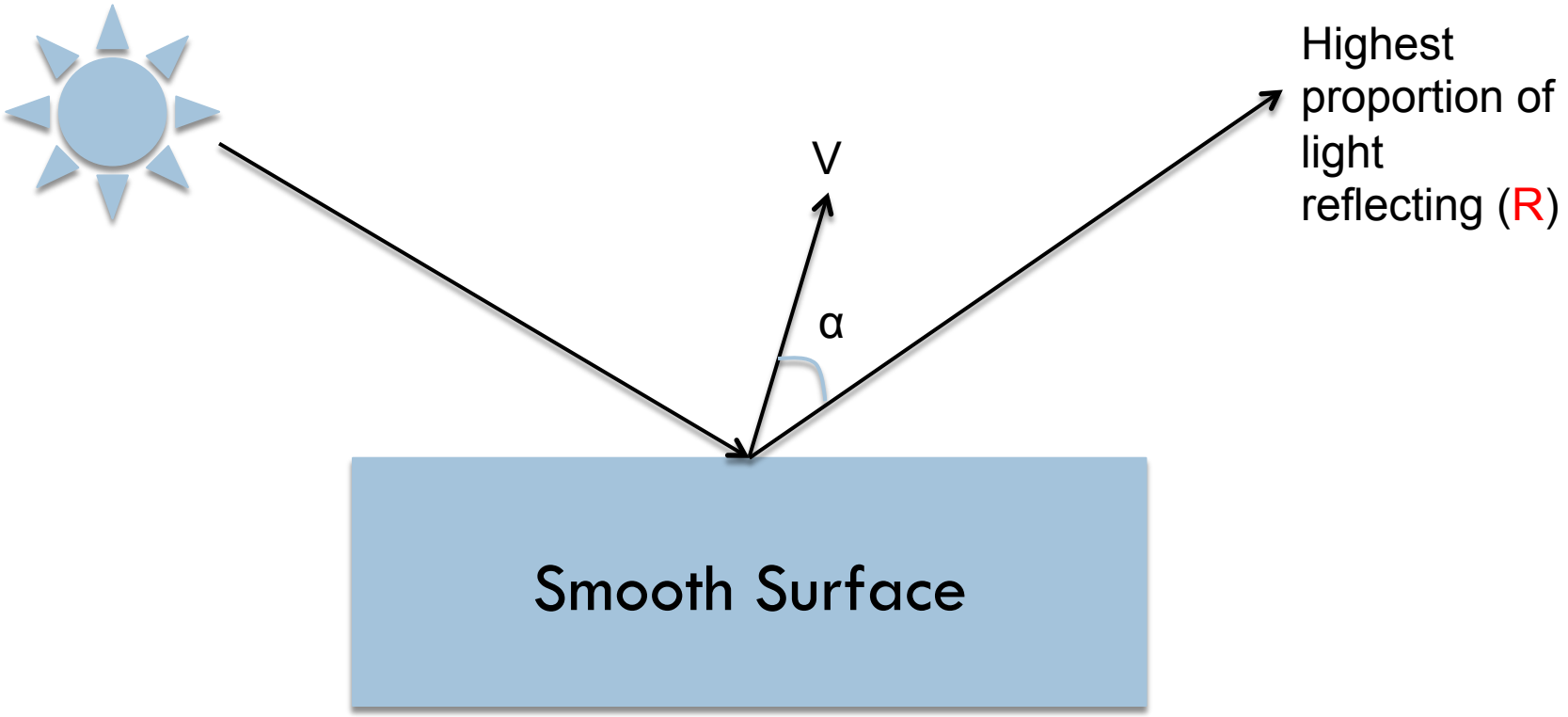
How much light reflects with specular lighting?



How much light gets to point V?

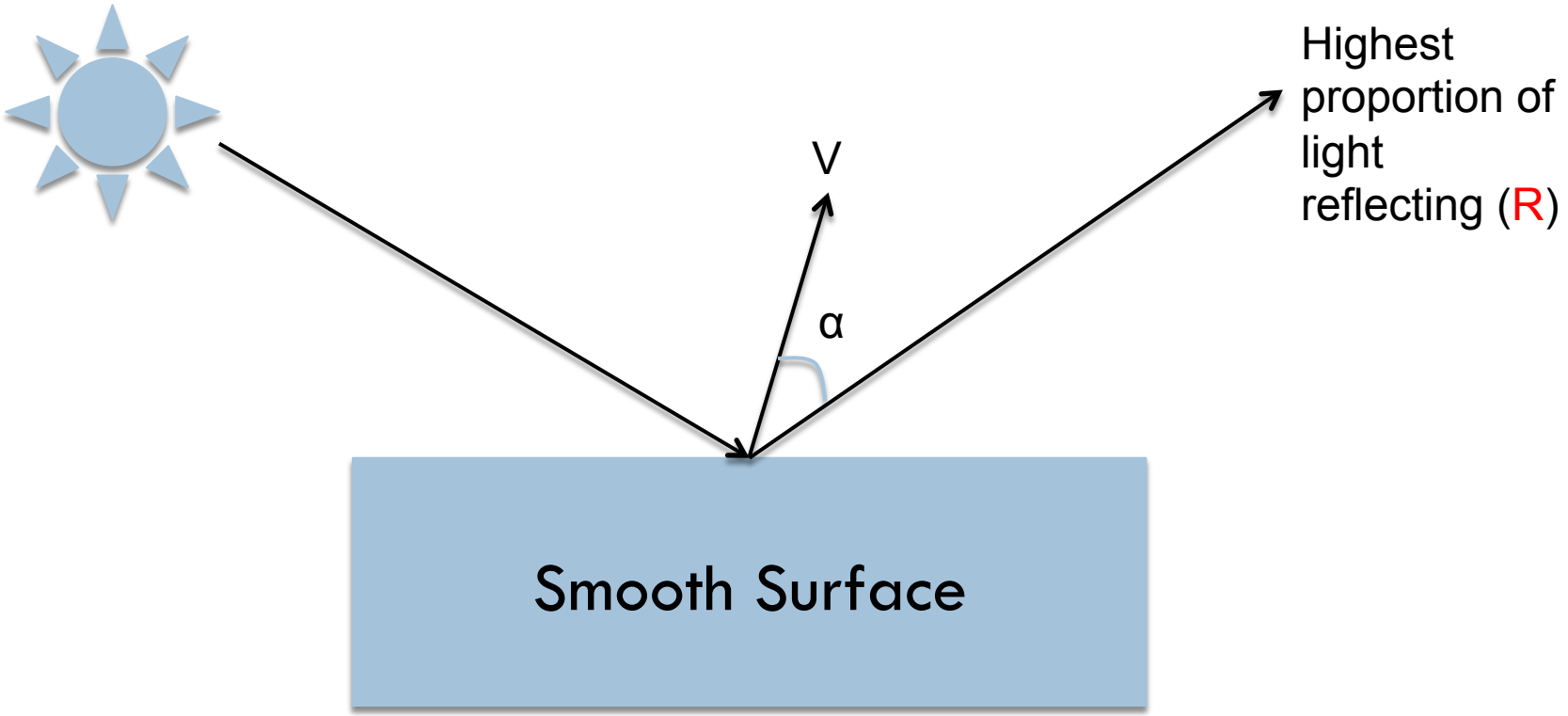
A: proportional to cos(α)
(Shininess strength) * cos(α) ^ (shininess coefficient)

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

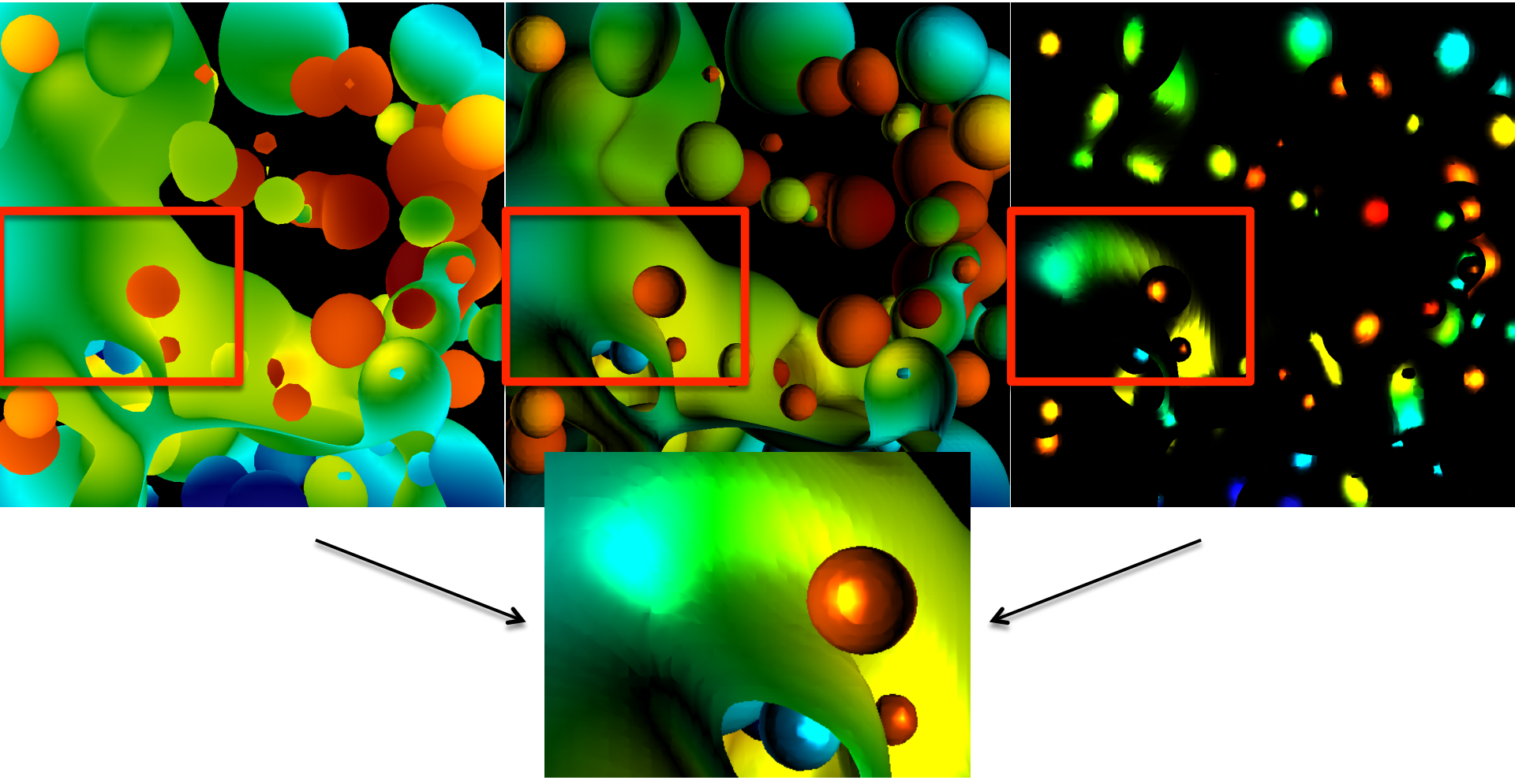
- For specular lighting, we will use one-sided lighting for project 1E
  - It just looks better
  - No "bounce back" light
  - Diffuse: abs(L·N)
  - Specular: max(0, S*(R·V)$^{\gamma}$)

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)
  - What should view direction be?

Context

- Projects 1B-1E all assume geometry has been transformed so that it overlaps with the screen
- The view direction (needed for many lighting equations) is lost / hard to calculate
- After the transformation, the view direction is close to (0, 0, -1)
  - We will use this value for 1E
    - Explicitly: use a view direction of (0, 0, -1) for all shading calculations in 1E

A Better Place to Calculate Shading

- If we have geometry in 3D space, then view direction makes sense
  - (Triangle vertex) minus (camera position)
- Then we can calculate shading then (in 3D space)
- And keep the shading value with us as we transform the triangle to overlap with the pixels on the screen
- This is how GPUs do it
- We will change our projects to do this more correct approach in 1F
  - And I may switch 1E and 1F in future terms

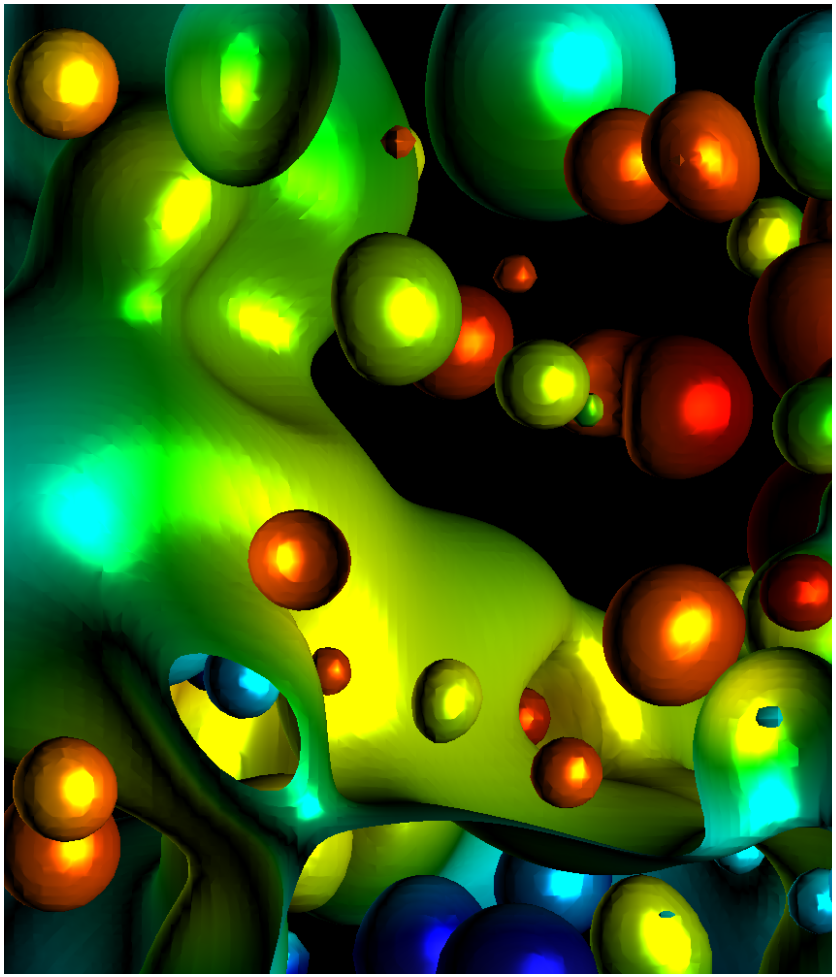
Lighting parameters

```
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 (6%) (due Oct 24th)

- 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];
};
```

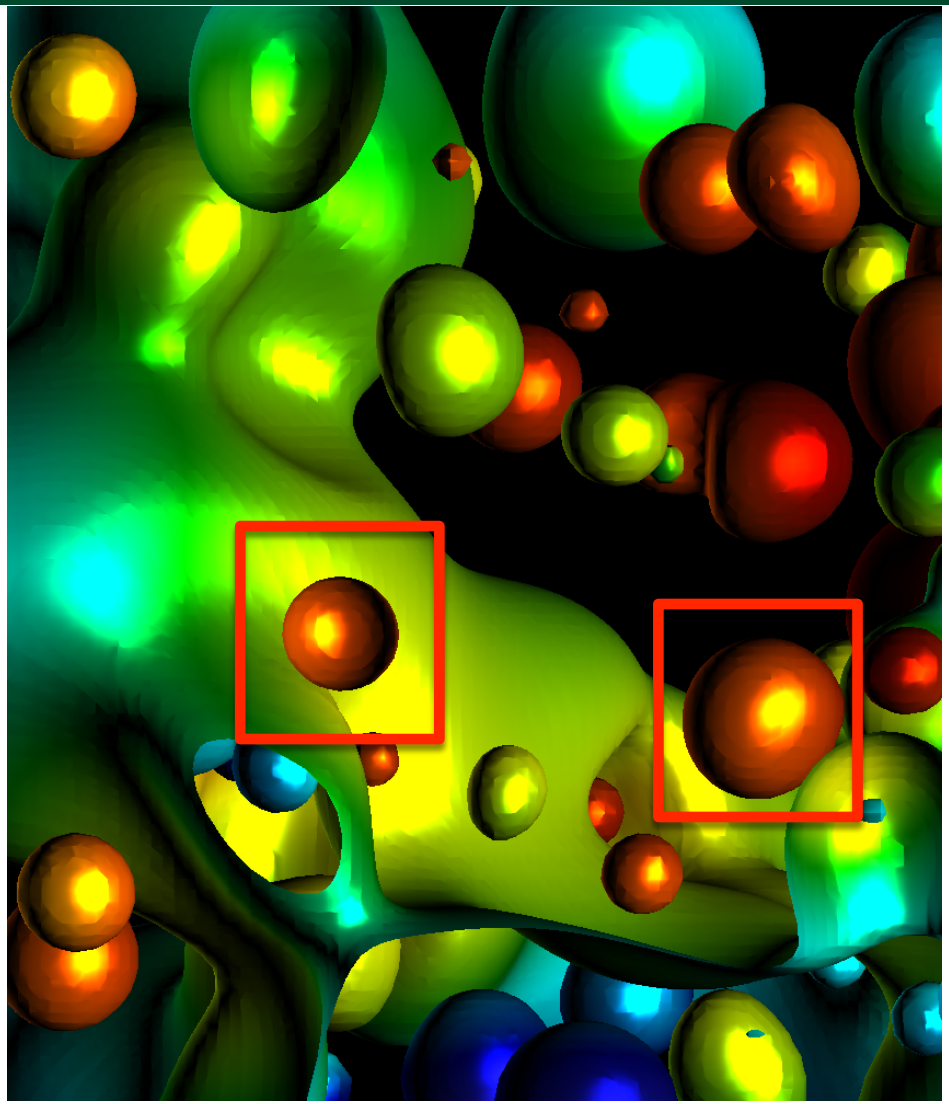
→reader1e.cxx will not compile until you make these changes

→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.

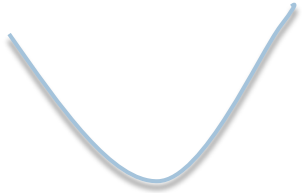
Where Hank spent his debugging time…



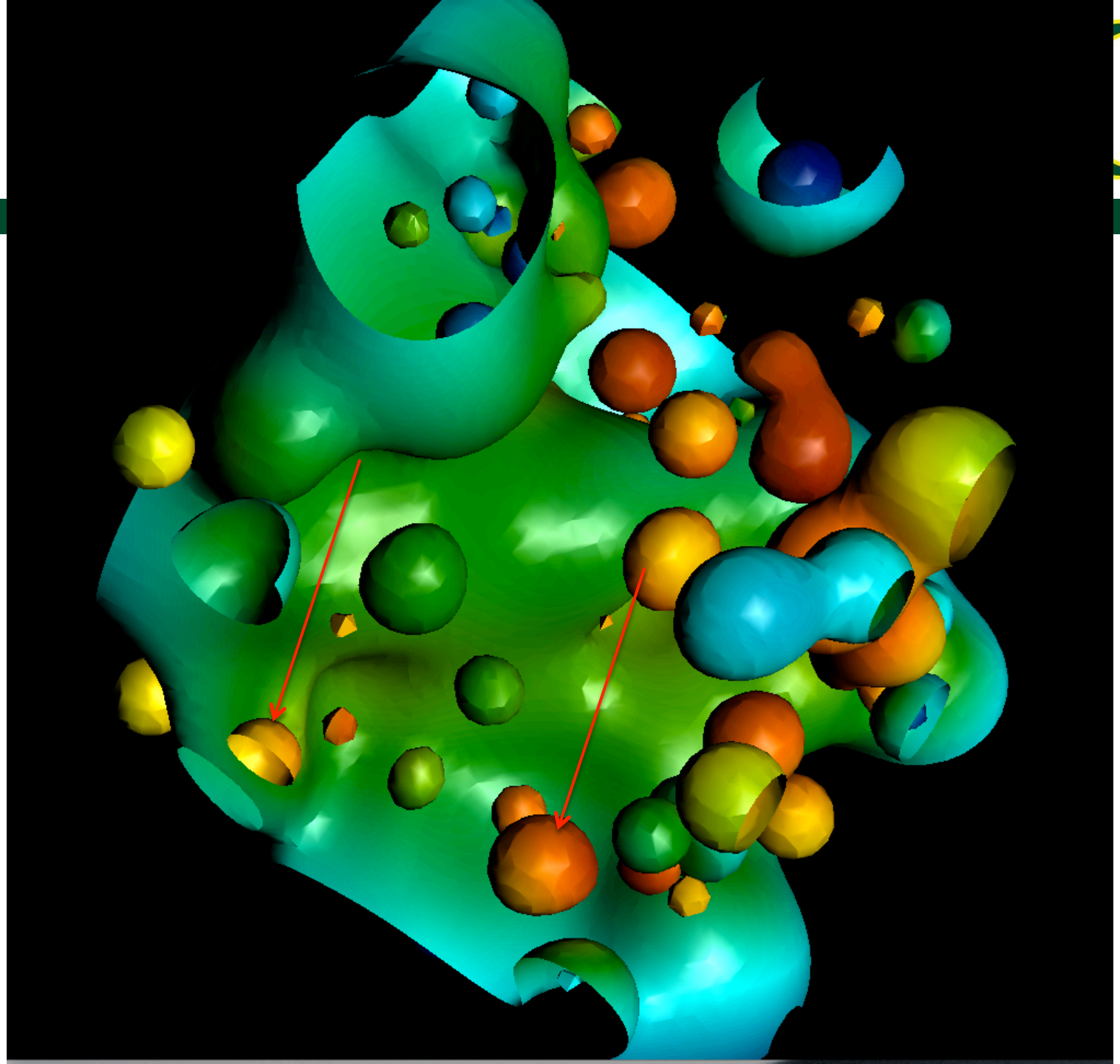
Concave surface

Lighting direction

Convex surface



Lighting direction



Now Let's Start On Arbitrary Camera Positions

Basic Transforms

Starting easy … two-dimensional

MP = P’
Matrix M transforms point P to make new point P’.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a*x + b*y \\ c*x + d*y \end{pmatrix}$$

M takes point (x,y)
to point (a*x+b*y, c*x+d*y)

MP = P’
Matrix M transforms point P to make new point P.

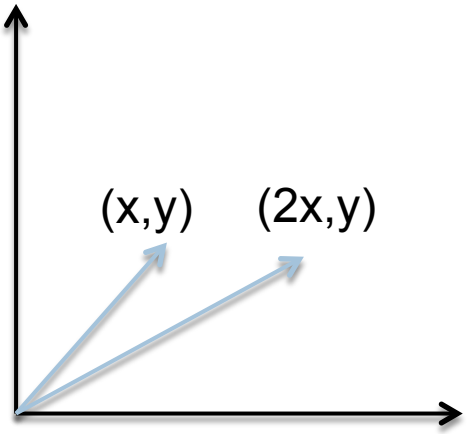
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

Identity Matrix

MP = P’

Matrix M transforms point P to make new point P.

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2x \\ y \end{pmatrix}$$

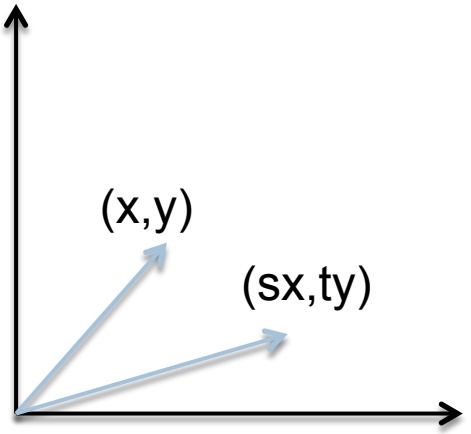


Scale in X, not in Y

MP = P’

Matrix M transforms point P to make new point P.

$$\begin{pmatrix} s & 0 \\ 0 & t \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} sx \\ ty \end{pmatrix}$$

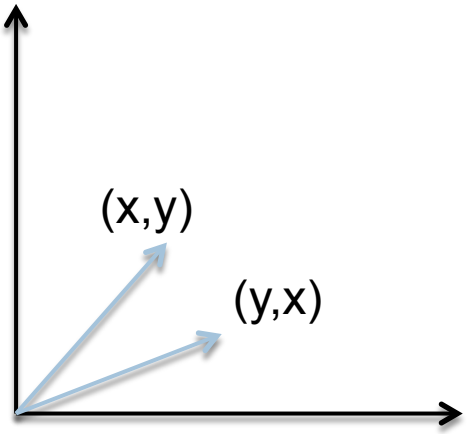


Scale in both dimensions

MP = P’

Matrix M transforms point P to make new point P.

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ x \end{pmatrix}$$

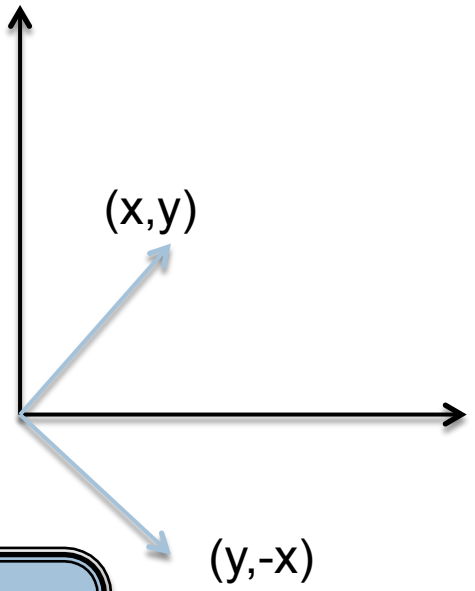


Switch X and Y

MP = P’

Matrix M transforms point P to make new point P.

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ -x \end{pmatrix}$$

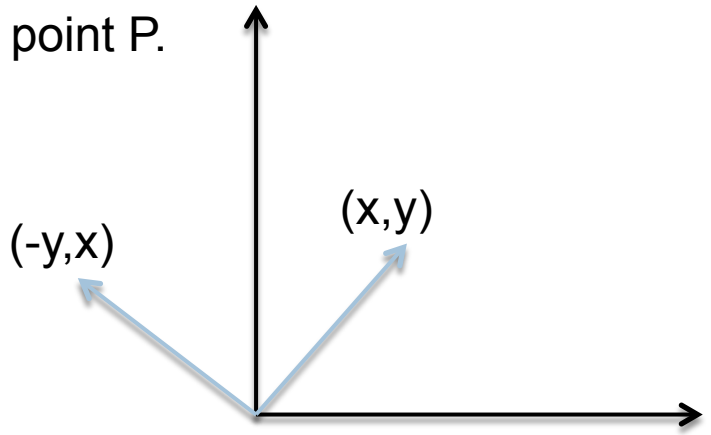


Rotate 90 degrees clockwise

MP = P'

Matrix M transforms point P to make new point P.

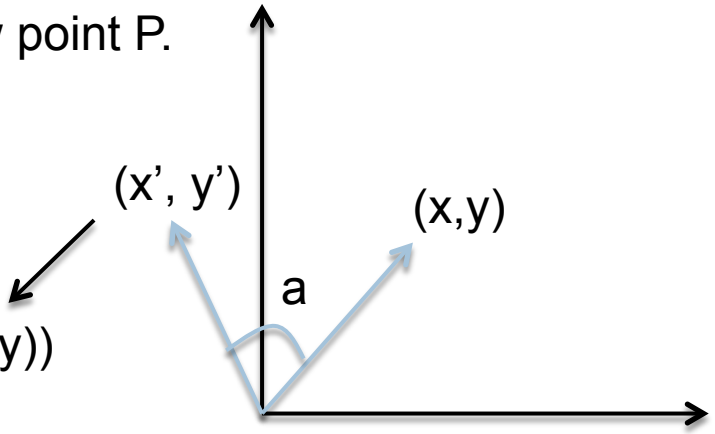
$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -y \\ x \end{pmatrix}$$



Rotate 90 degrees counter-clockwise

MP = P'

Matrix M transforms point P to make new point P.



$$\begin{pmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(a)*x+\sin(a)*y \\ -\sin(a)*x+\cos(a)*y \end{pmatrix}$$

Rotate "a" degrees counter-clockwise

Combining transformations

- How do we rotate by 90 degrees clockwise and then scale X by 2?
  - Answer: multiply by matrix that multiplies by 90 degrees clockwise, then multiple by matrix that scales X by 2.
  - But can we do this efficiently?

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -2 & 0 \end{pmatrix}$$

Reminder: matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a*e+b*g & a*f+b*h \\ c*e+d*g & c*f+d*h \end{pmatrix}$$

Combining transformations

- How do we rotate by 90 degrees clockwise and then scale X by 2?
  - Answer: multiply by matrix that rotates by 90 degrees clockwise, then multiply by matrix that scales X by 2.
  - But can we do this efficiently?

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -2 & 0 \end{pmatrix}$$

Combining transformations

- How do we scale X by 2 and then rotate by 90 degrees clockwise?
  - Answer: multiply by matrix that scales X by 2, then multiply by matrix that rotates 90 degrees clockwise.

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2 \\ -1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -2 & 0 \end{pmatrix}$$

Multiply then scale
Order matters!!

Translations

- Translation is harder:

$$\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a+c \\ b+d \end{pmatrix}$$

But this doesn't fit our nice matrix multiply model…
What to do??

Homogeneous Coordinates

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Add an extra dimension.
A math trick … don't overthink it.

Homogeneous Coordinates

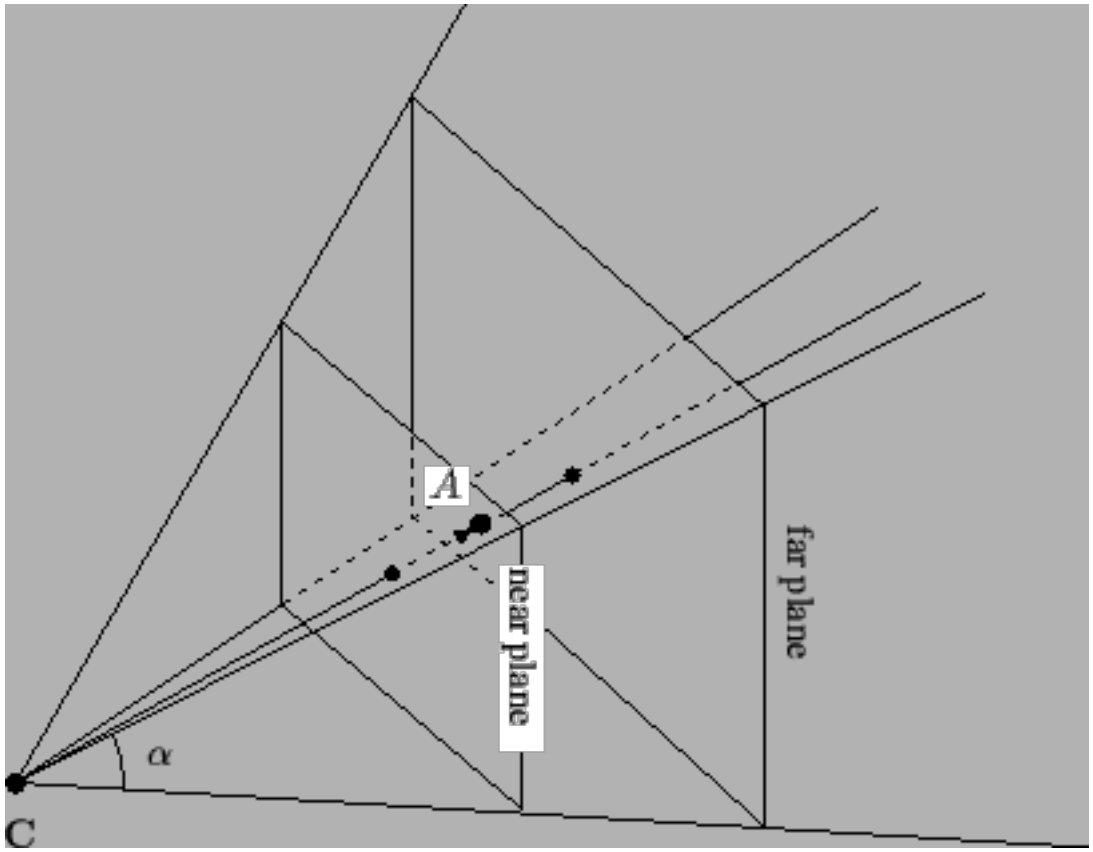
$$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+dx \\ y+dy \\ 1 \end{pmatrix}$$

Translation

We can now fit translation into
our matrix multiplication system.

Arbitrary Camera Positions

How do we specify a camera?



The “viewing pyramid” or “view frustum”.

Frustum: In geometry, a frustum (plural: frusta or frustums) is the portion of a solid (normally a cone or pyramid) that lies between two parallel planes cutting it.

```
class Camera
{
  public:
    double          near, far;
    double          angle;
    double          position[3];
    double          focus[3];
    double          up[3];
};
```

New terms

- Coordinate system:
  - A system that uses coordinates to establish position
- Example: (3, 4, 6) really means…
  - 3x(1,0,0)
  - 4x(0,1,0)
  - 6x(0,0,1)
- Since we assume the Cartesian coordinate system

New terms

- Frame:
  - A way to place a coordinate system into a specific location in a space
- Cartesian example: (3,4,6)
  - It is assumed that we are speaking in reference to the origin (location (0,0,0)).
- A frame F is a collection (v1, v2, …, vn, O) is a frame over a space if (v1, v2, … vn) form a basis over that space.
  - What is a basis?? ← linear algebra term

What does it mean to form a basis?

- For any vector v, there are unique coordinates (c1, …, cn) such that

  v = c1*v1 + c2*v2 + … + cn*vn

What does it mean to form a basis?

- For any vector v, there are unique coordinates (c1, …, cn) such that

  v = c1*v1 + c2*v2 + … + cn*vn

- Consider some point P.
  - The basis has an origin O
  - There is a vector v such that O+v = P
  - We know we can construct v using a combination of vi's
  - Therefore we can represent P in our frame using the coordinates (c1, c2, …, cn)

Example of Frames

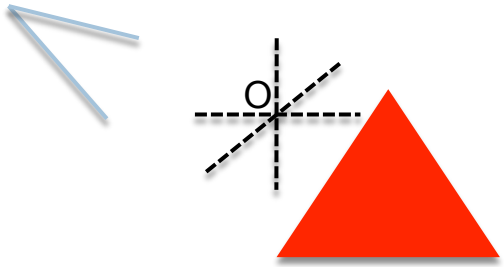
- Frame F = (v1, v2, O)
  - v1 = (0, -1)
  - v2 = (1, 0)
  - O = (3, 4)
- What are F's coordinates for the point (6, 6)?

Example of Frames

- Frame F = (v1, v2, O)
  - v1 = (0, -1)
  - v2 = (1, 0)
  - O = (3, 4)
- What are F's coordinates for the point (6, 6)?

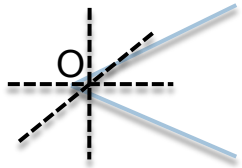
- Answer: (-2, 3)

Our goal



World space:

Triangles in native Cartesian coordinates
Camera located anywhere



Camera space:

Camera located at origin, looking down -Z
Triangle coordinates relative to camera frame

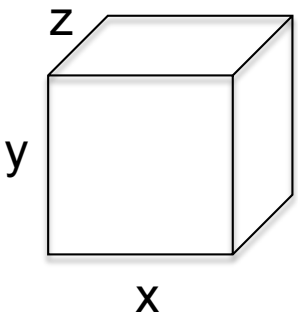
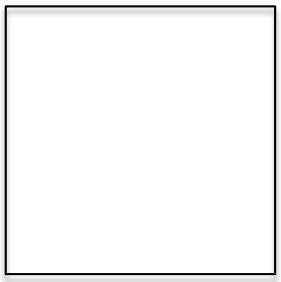


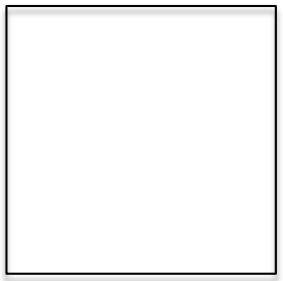
Image space:

All viewable objects within
-1 <= x,y,z <= +1



Screen space:

All viewable objects within
-1 <= x, y <= +1



Device space:

All viewable objects within
0<=x<=width, 0 <=y<=height