CS 543 - Computer Graphics: Ray Tracing Detail, Part 4

by

Robert W. Lindeman

gogo@wpi.edu

(with help from Emmanuel Agu ;-)

Reflection and Transparency

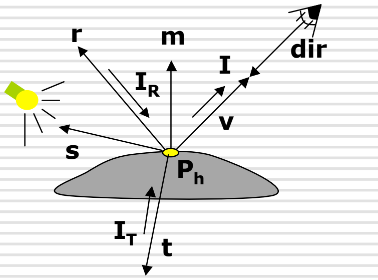
- □ Ray tracing also handles reflections and refraction of light well
- □ We can easily render realistic scenes with
  - mirrors
  - martini glasses
- □ So, far, we have considered **Local components** (ambient, diffuse, specular)
- □ Local components are contributions from light sources which are visible from hit point
- □ To render reflection, and refraction we need to add reflection and refraction components of light

$$I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + I_{tran}$$

Reflection and Transparency

- □ First three components are local

$$I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + I_{tran}$$

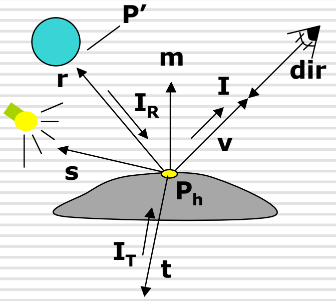


- □ Reflected component, I_R, is along mirror direction from eye **−r**

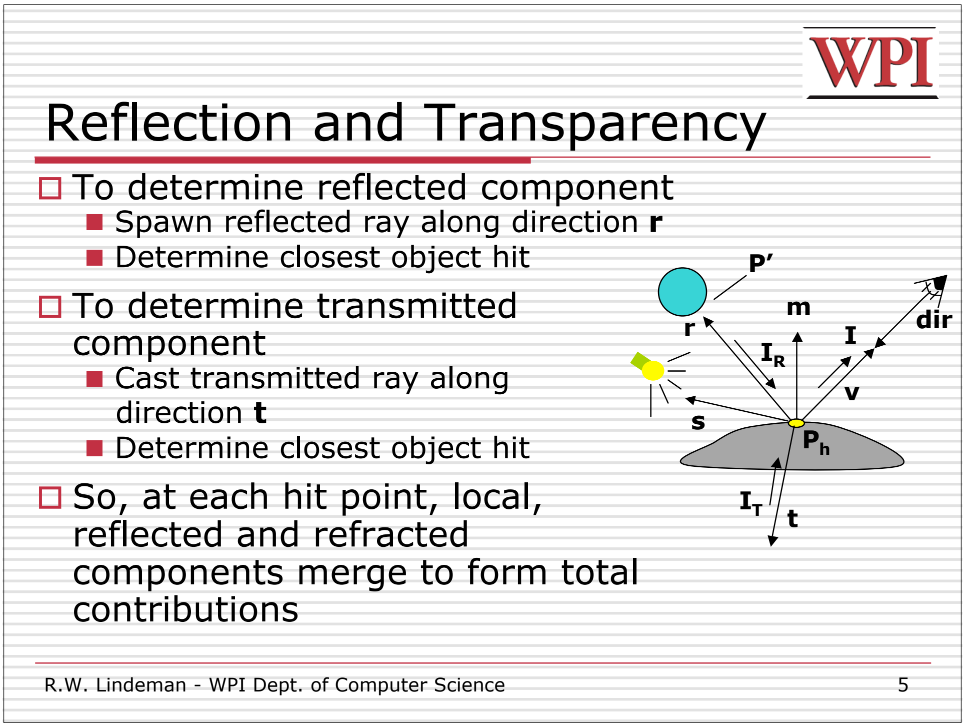
Reflection and Transparency

- □ **r** is given as (see eqn 4.22) as

$$\mathbf{r} = \mathbf{dir} - 2(\mathbf{dir} \bullet \mathbf{m})\mathbf{m}$$



- □ Transmitted component I_T is along transmitted direction **t**
- □ Portion of light coming in from direction **t** is bent along **dir**
- □ I_R and I_T each have their own five components (ambient, diffuse, etc)
- □ In some sense, point P′ along reflected direction **r** serves as a light source to point P_h

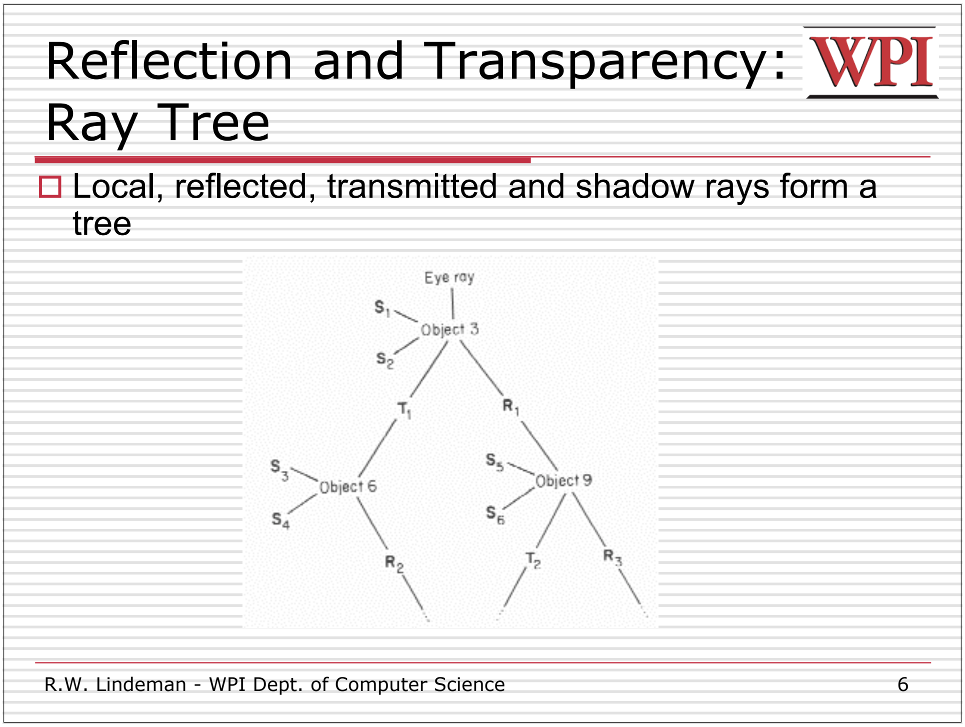


Reflection and Transparency

WPI

- □ To determine reflected component
  - Spawn reflected ray along direction **r**
  - Determine closest object hit
- □ To determine transmitted component
  - Cast transmitted ray along direction **t**
  - Determine closest object hit
- □ So, at each hit point, local, reflected and refracted components merge to form total contributions

P′
m
r
I_R
I
dir
v
s
P_h
I_T
t

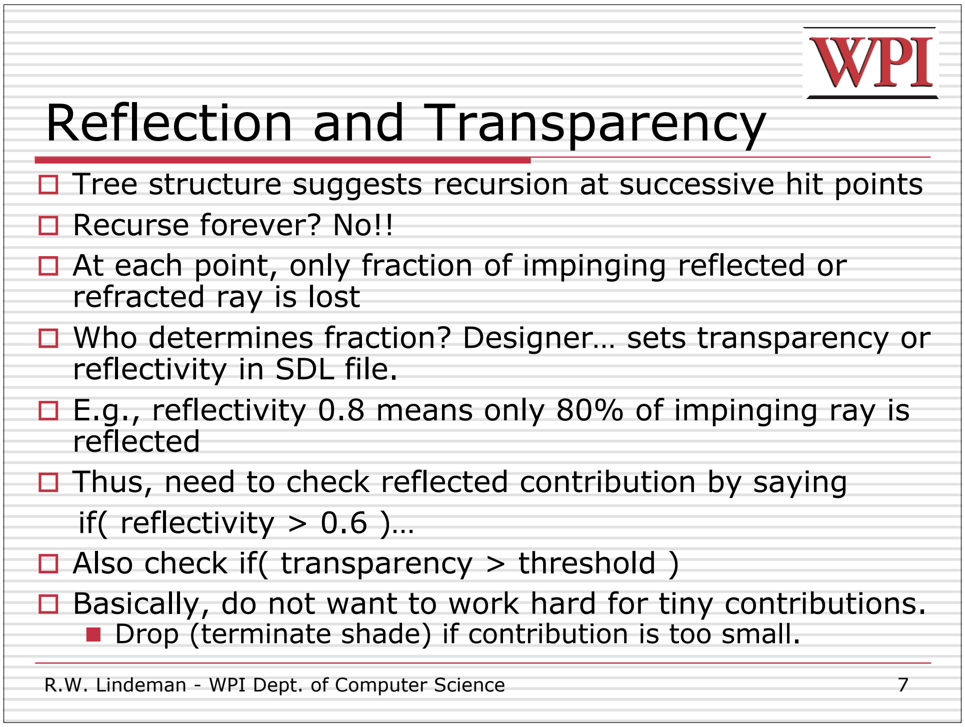


Reflection and Transparency: Ray Tree

WPI

- □ Local, reflected, transmitted and shadow rays form a tree

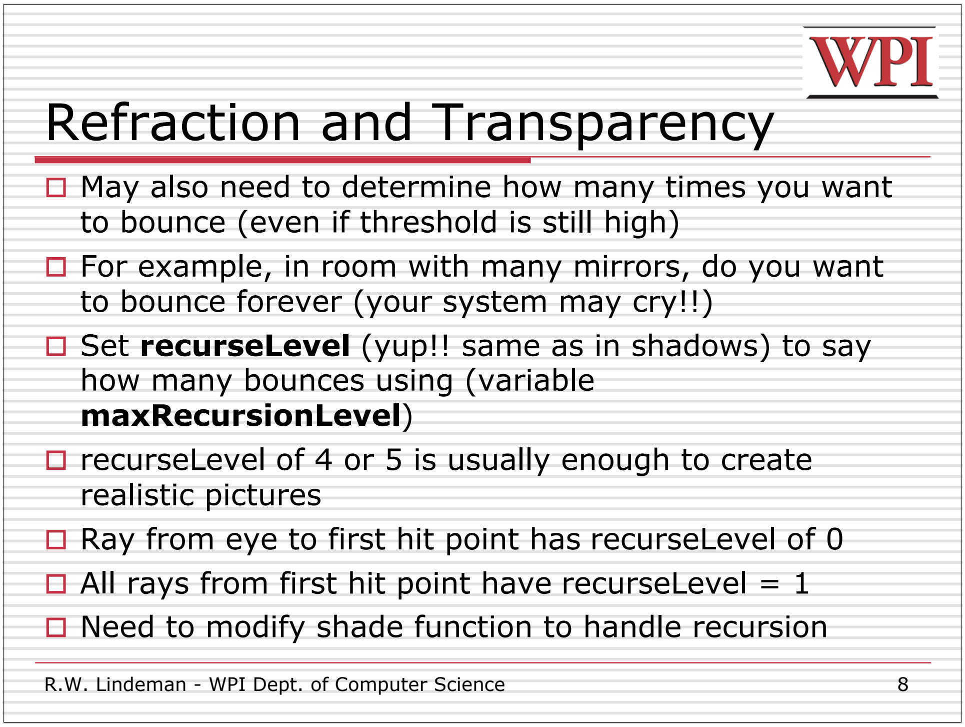
Eye ray
S_1
Object 3
S_2
T_1
R_1
S_3
Object 6
S_4
S_5
Object 9
S_6
R_2
T_2
R_3



Reflection and Transparency

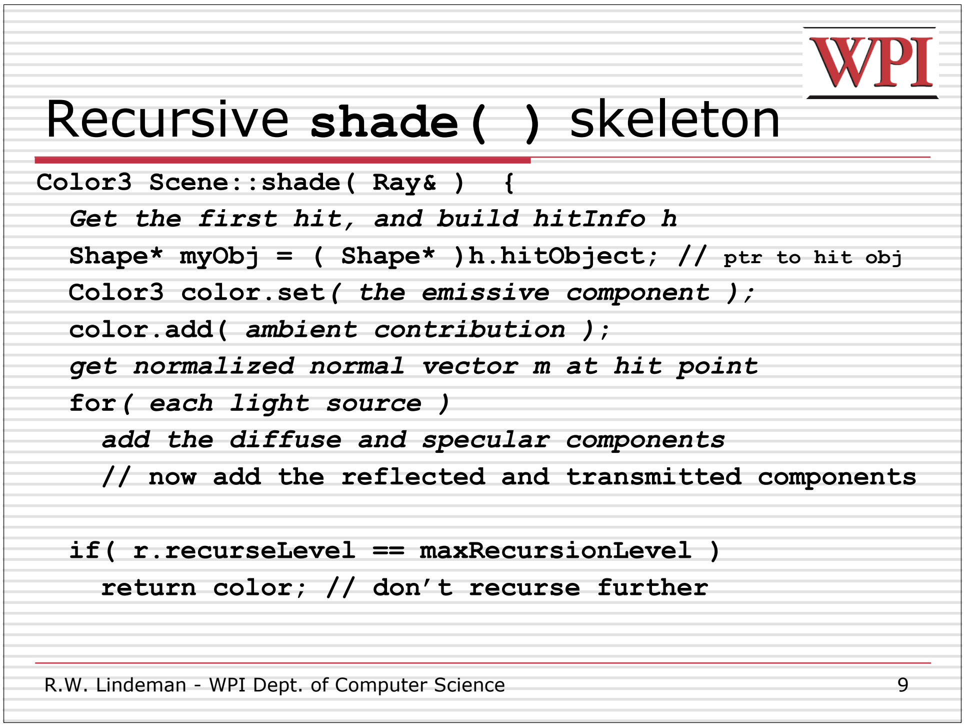
- Tree structure suggests recursion at successive hit points
- Recurse forever? No!!
- At each point, only fraction of impinging reflected or refracted ray is lost
- Who determines fraction? Designer… sets transparency or reflectivity in SDL file.
- E.g., reflectivity 0.8 means only 80% of impinging ray is reflected
- Thus, need to check reflected contribution by saying

  if(reflectivity > 0.6)…
- Also check if(transparency > threshold)
- Basically, do not want to work hard for tiny contributions.
  - Drop (terminate shade) if contribution is too small.



Refraction and Transparency

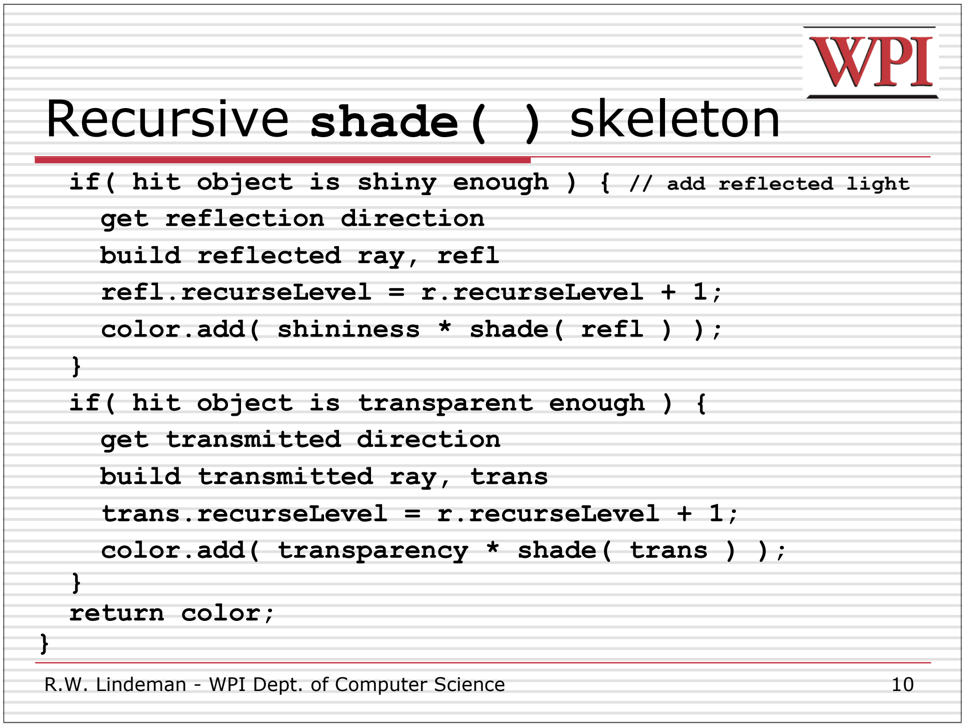
- May also need to determine how many times you want to bounce (even if threshold is still high)
- For example, in room with many mirrors, do you want to bounce forever (your system may cry!!)
- Set **recurseLevel** (yup!! same as in shadows) to say how many bounces using (variable **maxRecursionLevel**)
- recurseLevel of 4 or 5 is usually enough to create realistic pictures
- Ray from eye to first hit point has recurseLevel of 0
- All rays from first hit point have recurseLevel = 1
- Need to modify shade function to handle recursion



Recursive **shade( )** skeleton

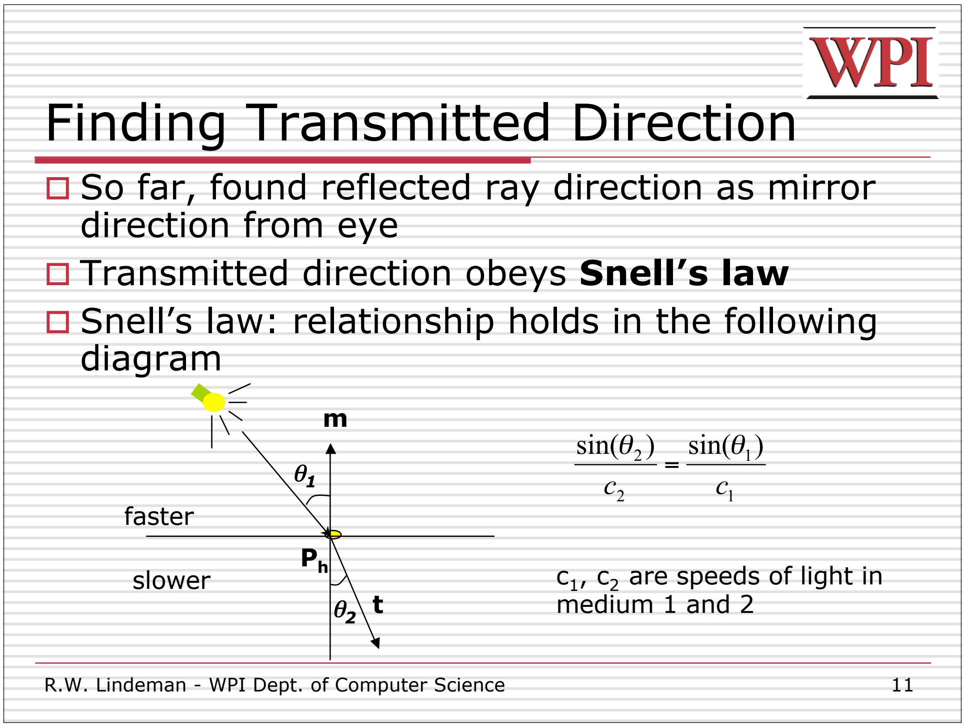
```
Color3 Scene::shade( Ray& )  {
  Get the first hit, and build hitInfo h
  Shape* myObj = ( Shape* )h.hitObject; // ptr to hit obj
  Color3 color.set( the emissive component );
  color.add( ambient contribution );
  get normalized normal vector m at hit point
  for( each light source )
    add the diffuse and specular components
    // now add the reflected and transmitted components

  if( r.recurseLevel == maxRecursionLevel )
    return color; // don't recurse further
```



Recursive **shade( )** skeleton

```
  if( hit object is shiny enough ) { // add reflected light
    get reflection direction
    build reflected ray, refl
    refl.recurseLevel = r.recurseLevel + 1;
    color.add( shininess * shade( refl ) );
  }
  if( hit object is transparent enough ) {
    get transmitted direction
    build transmitted ray, trans
    trans.recurseLevel = r.recurseLevel + 1;
    color.add( transparency * shade( trans ) );
  }
  return color;
}
```

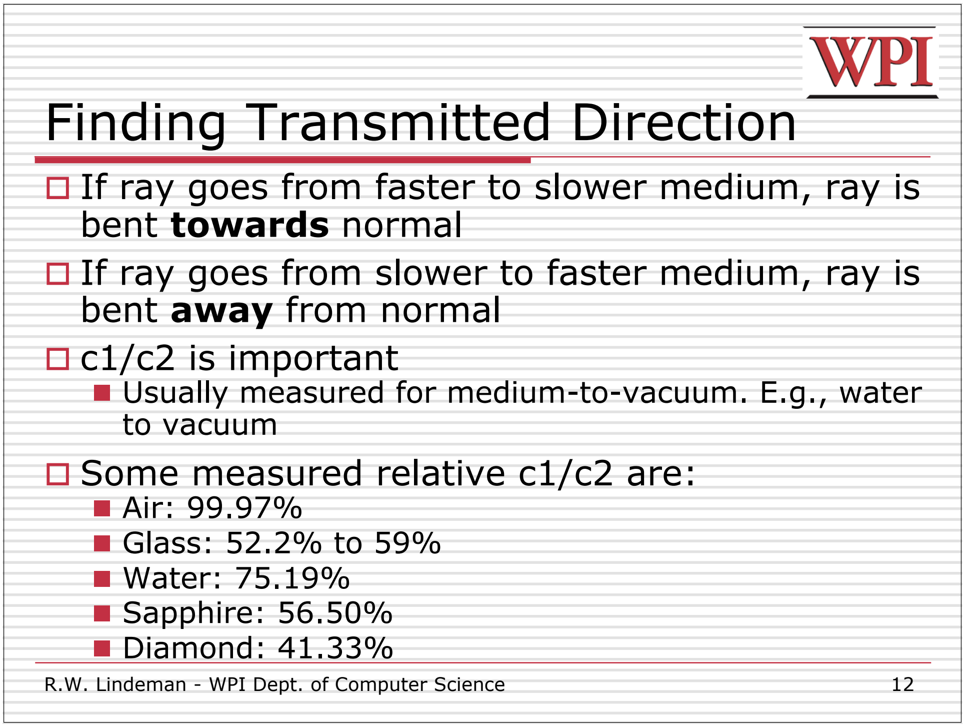


Finding Transmitted Direction

- □ So far, found reflected ray direction as mirror direction from eye
- □ Transmitted direction obeys **Snell's law**
- □ Snell's law: relationship holds in the following diagram

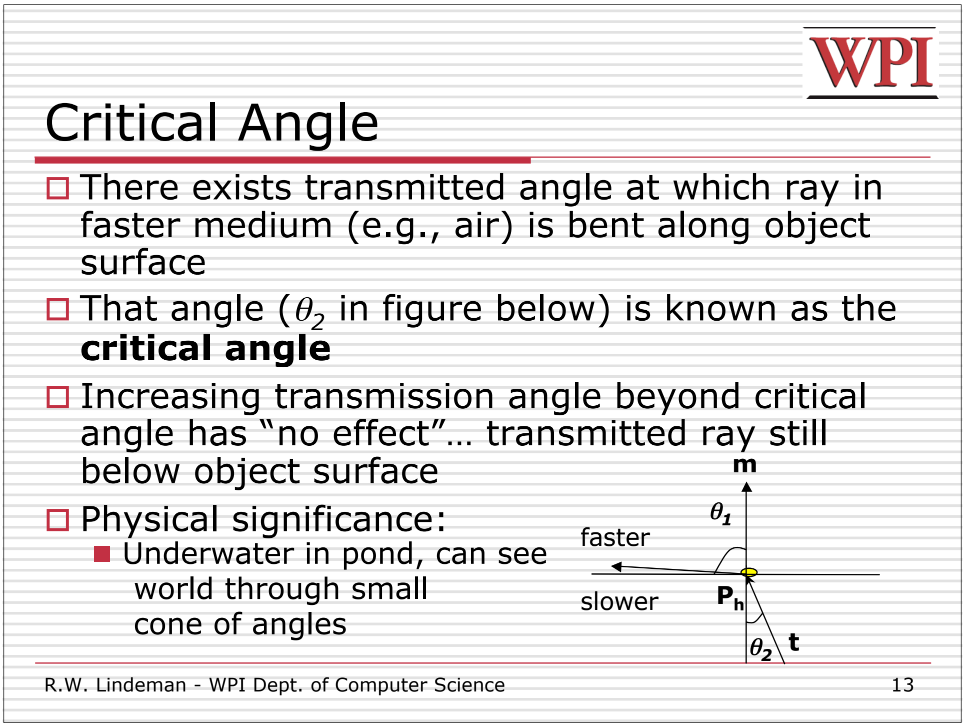
$$\frac{\sin(\theta_2)}{c_2} = \frac{\sin(\theta_1)}{c_1}$$

c_1, c_2 are speeds of light in medium 1 and 2



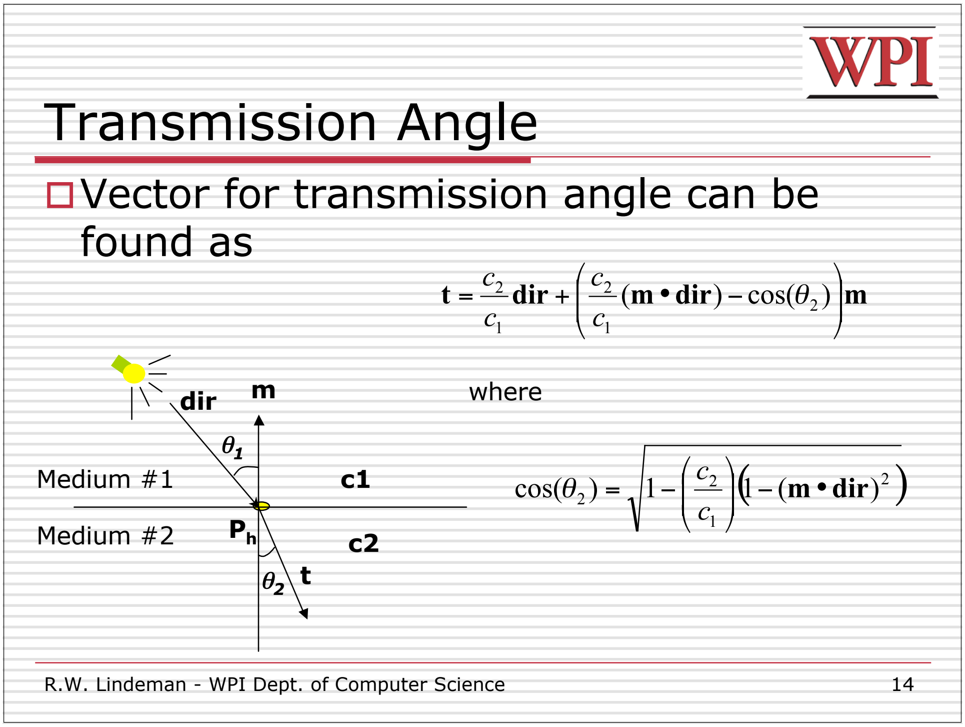
Finding Transmitted Direction

- □ If ray goes from faster to slower medium, ray is bent **towards** normal
- □ If ray goes from slower to faster medium, ray is bent **away** from normal
- □ c1/c2 is important
  - ■ Usually measured for medium-to-vacuum. E.g., water to vacuum
- □ Some measured relative c1/c2 are:
  - ■ Air: 99.97%
  - ■ Glass: 52.2% to 59%
  - ■ Water: 75.19%
  - ■ Sapphire: 56.50%
  - ■ Diamond: 41.33%



Critical Angle

- □ There exists transmitted angle at which ray in faster medium (e.g., air) is bent along object surface
- □ That angle (θ_2 in figure below) is known as the **critical angle**
- □ Increasing transmission angle beyond critical angle has "no effect"… transmitted ray still below object surface
- □ Physical significance:
  - Underwater in pond, can see world through small cone of angles



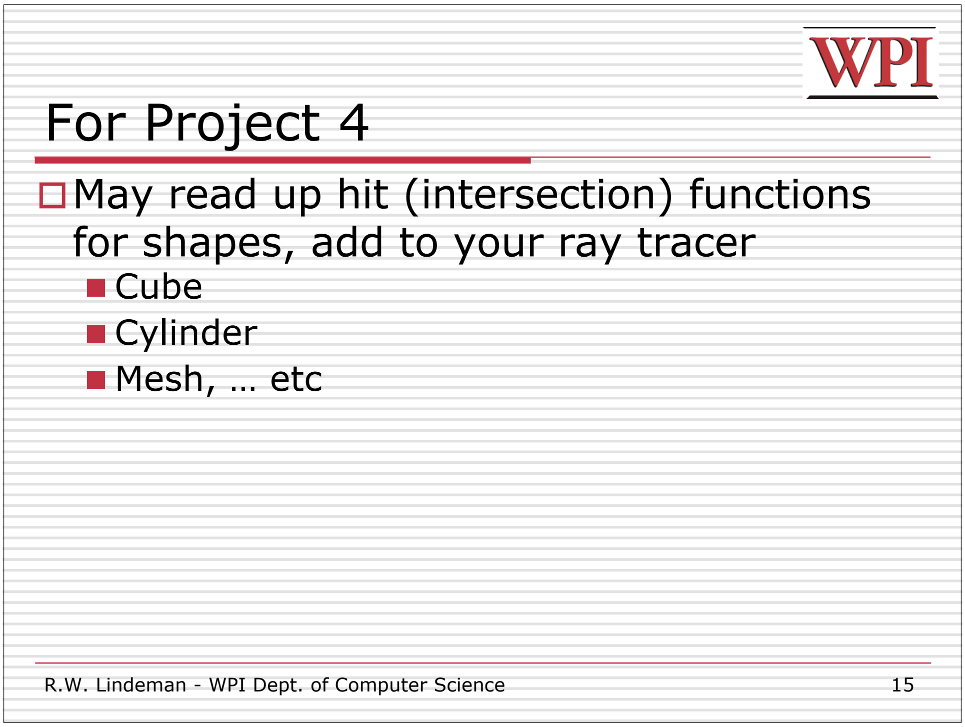
Transmission Angle

- □ Vector for transmission angle can be found as

$$\mathbf{t} = \frac{c_2}{c_1}\mathbf{dir} + \left(\frac{c_2}{c_1}(\mathbf{m} \bullet \mathbf{dir}) - \cos(\theta_2)\right)\mathbf{m}$$

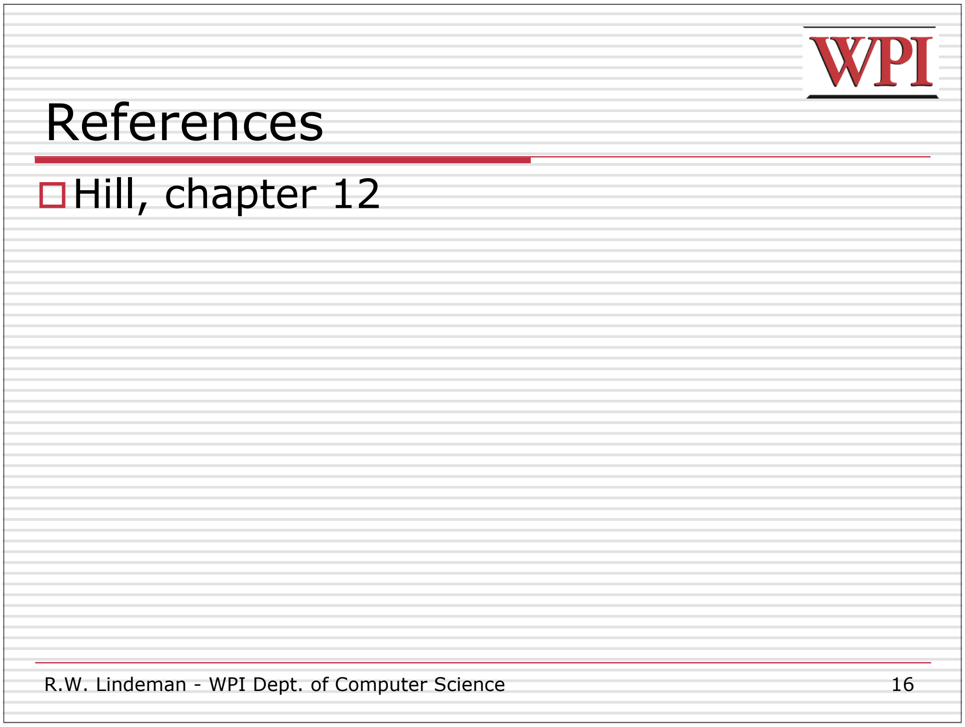
where

$$\cos(\theta_2) = \sqrt{1 - \left(\frac{c_2}{c_1}\right)\left(1 - (\mathbf{m} \bullet \mathbf{dir})^2\right)}$$



For Project 4

- May read up hit (intersection) functions for shapes, add to your ray tracer
  - Cube
  - Cylinder
  - Mesh, … etc



References

- Hill, chapter 12