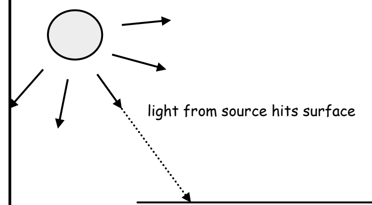
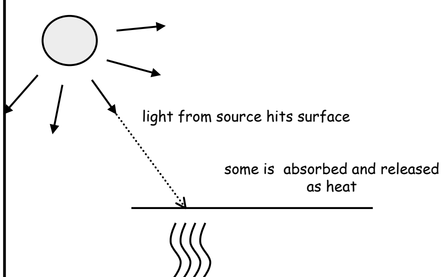
color

for each channel we'll approximate the color at the intersection point as the sum of five terms
- emission
- ambient reflection
- diffuse reflection
- specular reflection
- **specular transmission**

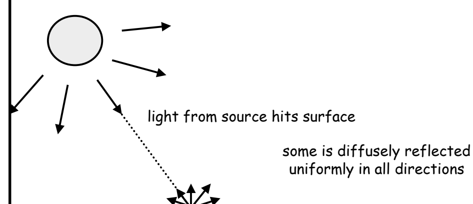
first a recap



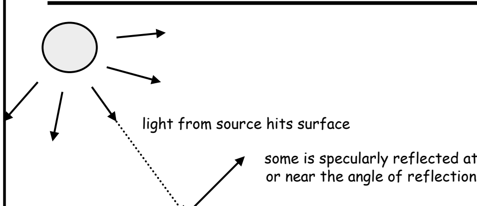
first a recap



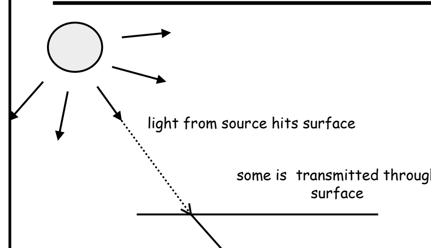
first a recap



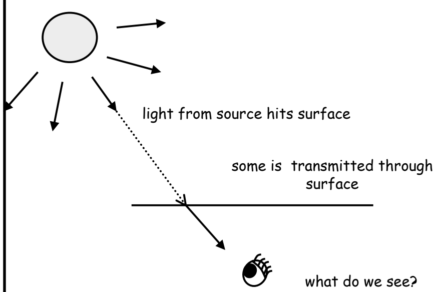
first a recap



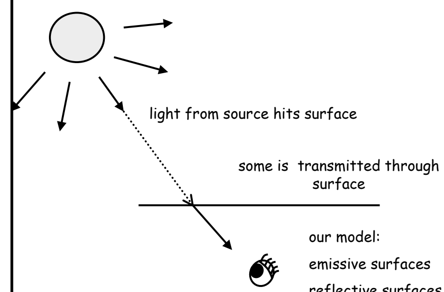
first a recap



transmission



transmission

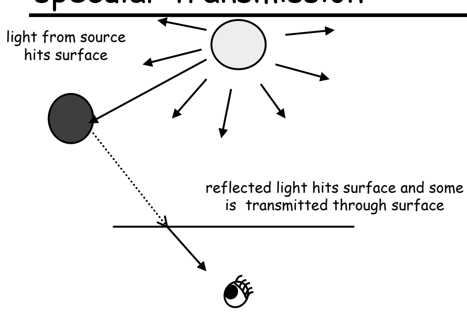


our model:

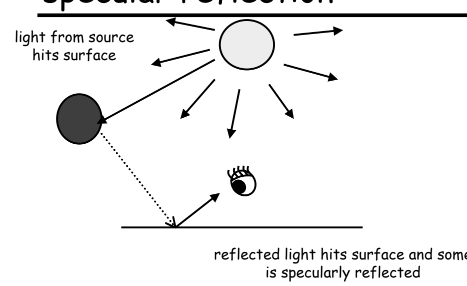
emissive surfaces

reflective surfaces lit by directional, point and spot lights that

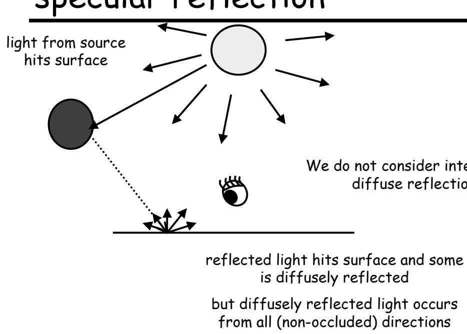
specular transmission



specular reflection

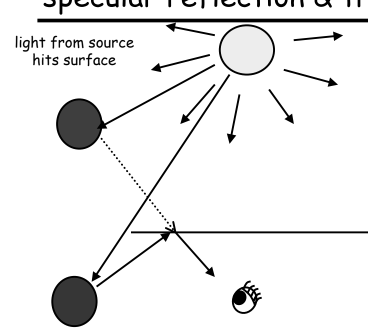


specular reflection



We do not consider inter-object diffuse reflections

specular reflection & transmission



We will recursively compute specular reflection and transmission

color: simple ray casting

for each channel we'll approximate the color at the intersection point as the sum of five terms

- emission
- ambient reflection
- diffuse reflection
- specular reflection
- ~~**specular transmission**~~ no contribution

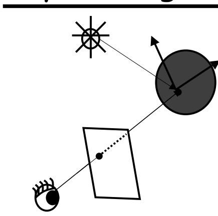
ray tracing

- simple ray casting
- **recursive ray tracing**
- cheap tricks
- optimizations

global effects

- shadows
- specular reflection
- specular transmission

ray tracing



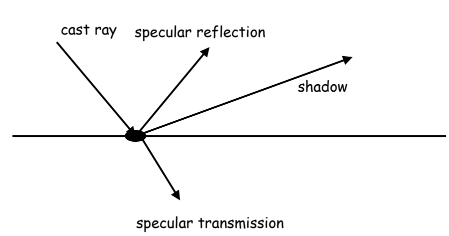
- cast ray through pixel into scene
- find closest intersection (if any)
- compute luminance at intersection
  - direct illumination (**no occlusions**)
  - **indirect reflection**

color: ray tracing

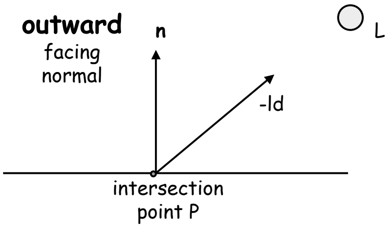
for each channel we'll approximate the color at the intersection point as the sum of five terms

- emission
- ambient reflection
- diffuse reflection **(check for shadows)**
- specular reflection **(check for shadows)**
- **recursive term**

recursive rays

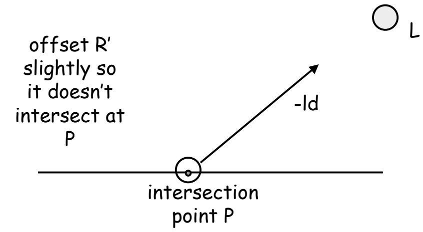


occlusion (shadows)



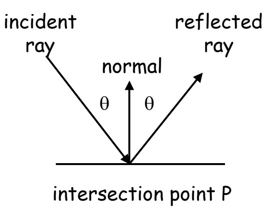
light L is occluded if the ray R'=(P,-ld) intersects some object in the scene

occlusion: implementation



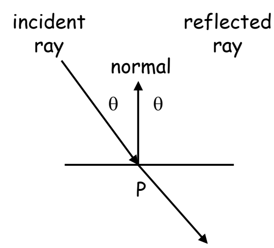
L is occluded if the ray (P,-ld) intersects some object in the scene

reflections



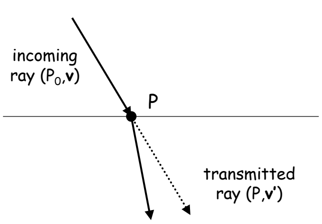
- cast ray reflected at P into scene
- find closest intersection point P' (if any)
- compute color at P'
- scale by (1-k_{trans})* msr[g,b] and add to color at P

transmission



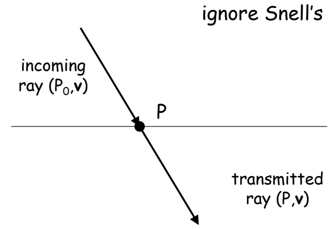
- cast ray transmitted at P into scene
- find closest intersection point P' (if any)
- compute color at P'
- scale by k_{trans}*msr[g,b] and add to color at P

refraction – snell's law

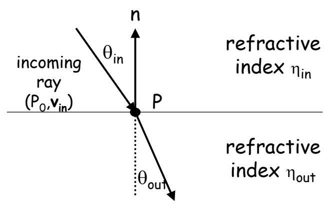


thin surface refraction

ignore Snell's law



thick surface refraction



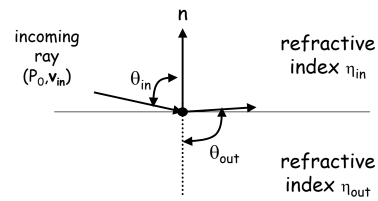
θ_{out} satisfies: $\eta_{out} \sin \theta_{out} = \eta_{in} \sin \theta_{in}$

$\Updownarrow$

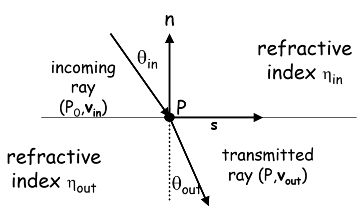
$\sin \theta_{out} = \beta \sin \theta_{in}$ where $\beta = \eta_{in}/\eta_{out}$
provided $0 \leq \beta \sin \theta_{in} \leq 1$

What if $\beta\sin\theta_{in} < 1$?

$\theta_{out} > 90°$: no transmission



thick surface refraction



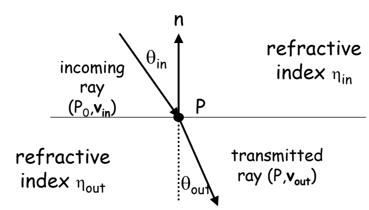
$\mathbf{v}_{out} = \cos \theta_{out}\, (\mathbf{-n}) + \sin \theta_{out}\, \mathbf{s}$

$\sin \theta_{out} = \beta \sin \theta_{in}$

$\cos \theta_{out} = (1-\sin^2 \theta_{out})^{1/2} = (1-\beta^2 \sin^2 \theta_{in})^{1/2}$

$\mathbf{s} = \mathbf{v}_{in} - (\mathbf{-n \bullet v}_{in})\, \mathbf{n} = \mathbf{v}_{in} + \cos(\theta_{in})\, \mathbf{n}$

thick surface refraction



θ_{out} satisfies: $\eta_{out} \sin \theta_{out} = \eta_{in} \sin \theta_{in}$

if $1- \beta^2 \sin^2 \theta_{in} \geq 0$ where $\beta = \eta_{in}/\eta_{out}$ then

$\mathbf{v}_{out} = [\beta \cos \theta_{in} - (1- \beta^2 \sin^2 \theta_{in})^{\frac{1}{2}}]\, \mathbf{n} + \beta \mathbf{v}_{in}$

stopping conditions

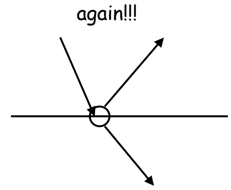
recurse until:

a) maximum recursive depth specified by user is reached
b) contribution to color is less than user specified bound

- cast new ray from P into scene
- find closest intersection point P' (if any)
- compute color at P'
- scale and add to color at P

implementation issues

offset new ray slightly to make sure you don't find P again!!!



- cast new ray from P into scene
- find **closest** intersection point P' (if any)
- compute color at P'
- scale and add to color at P

ray tracing

- simple ray casting
- recursive ray tracing
- **cheap tricks**
- optimizations

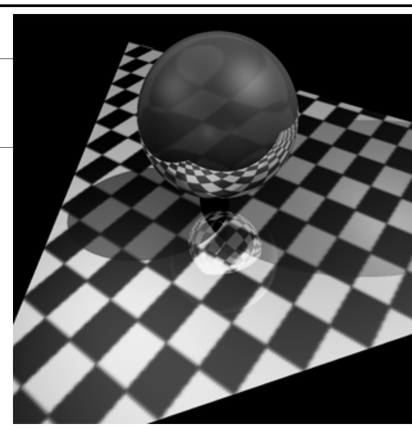
cheap tricks

- texture mapping
- procedural texture mapping
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

texture mapping

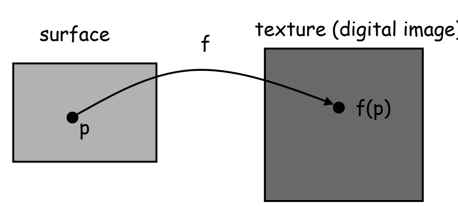
"glue" image to surface

steve yan,
fall 2001



drew levin,
fall 2001

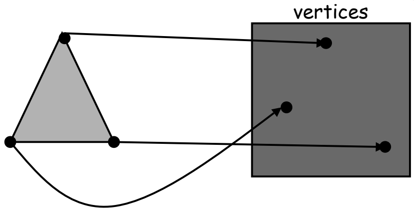
texture mapping



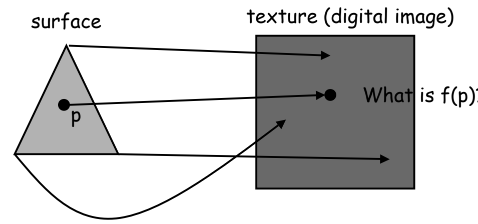
f(p): coordinates in the texture map
corresponding to surface point p

texture mapping triangle

Input specifies texture coordinates of triangle vertices

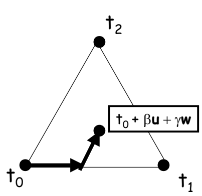


texture mapping triangle

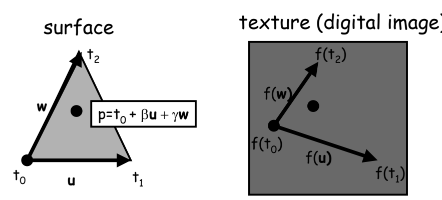


triangle: parametric form

a point q on the triangle T can be uniquely represented as $q=t_0+\beta\mathbf{u}+\gamma\mathbf{w}$ where $\beta\geq 0$, $\gamma\geq 0$, $\beta+\gamma\leq 1$

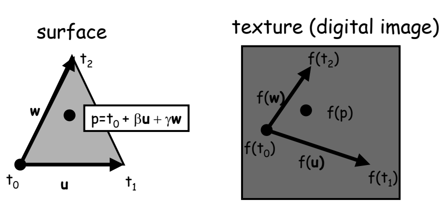


computing f(p)



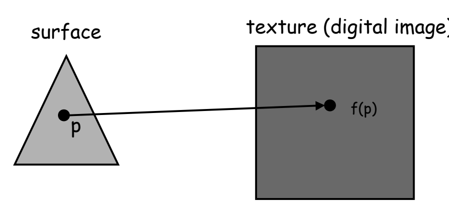
compute $f(\mathbf{u}) = f(t_1)-f(t_0)$ and $f(\mathbf{w})=f(t_2)-f(t_0)$

computing f(p)



compute $f(p) = f(t_0)+\beta f(\mathbf{u})+\gamma f(\mathbf{w})$

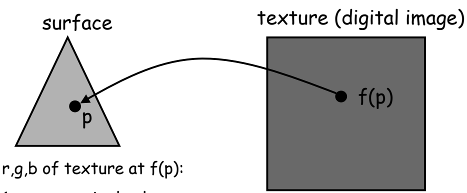
texture mapping triangle



What is image color at f(p)?

Need to resample! For your ray tracer use bilinear interpolation.

using the color



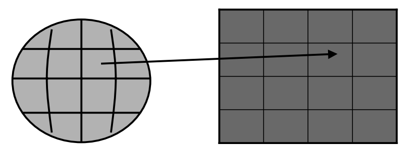
r,g,b of texture at f(p):

1. use as pixel color
2. use as diffuse and specular coefficient of surface at p
3. use as diffuse coefficient of surface at p

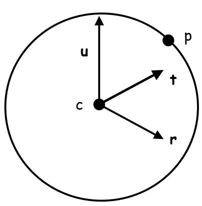
DO THIS!

texture mapping sphere

e.g. latitude/longitude

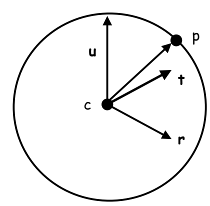


parameterization



a point p on the surface can be represented relative c and **r**, **t**, **u**

parameterization



p-c = α**r** + β**t** + γ**u**

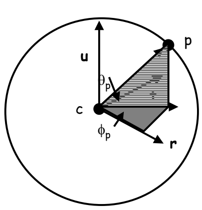
where

α=

β=

γ=

parameterization



p-c = α**r** + β**t** + γ**u**

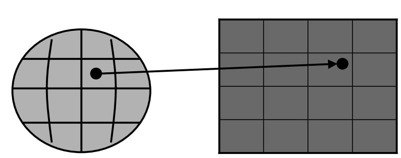
where

α= r $\cos(\theta_p)\cos(\phi_p)$

β= r $\cos(\theta_p)\sin(\phi_p)$

γ= r $\sin\theta_p$

texture mapping sphere



f(p) = ($w\phi_p$ /360, $h\theta_p$ /360) where the the angles are in degrees and w, h are the image width and height in pixels

as before

- resample image
- use color as ...

problems

- given p compute ϕ, θ

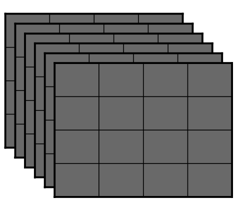
you can do that!

- poles

test for pole and use default texture coordinate

- seams
  - use good textures
  - overlap & blend or mix
  - don't look there
  - 3d textures

3d textures



use stack of images

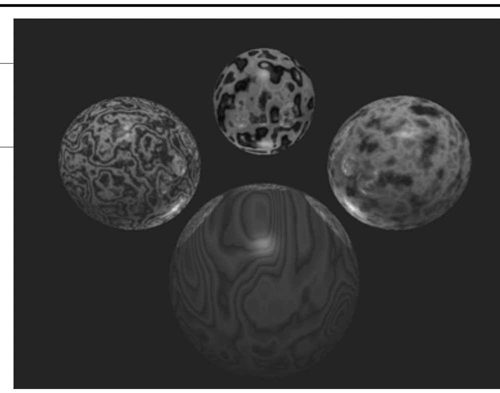
how do we generate these images?

cheap tricks

- texture mapping
- **procedural texture mapping**
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

procedural texture mapping

- procedure returns a texture color for any point in 3d space (note this is not an image stack)
- sample to find texture for surface



adrian mettler, spring 2003

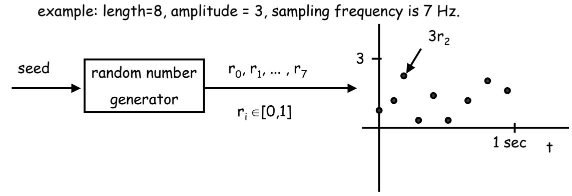
procedural textures

- advantages
  - don't need to find a mapping from a (complex) 3d surface to a 2d texture image
  - concise representation of texture
- disadvantages
  - ad hoc techniques cannot duplicate photographs

perlin noise – 1D example

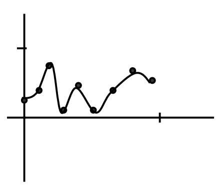
step 1: generate discrete noise function with specified length, amplitude, sampling frequency

example: length=8, amplitude = 3, sampling frequency is 7 Hz.



perlin noise – 1D example

step 2: interpolate with smoothing

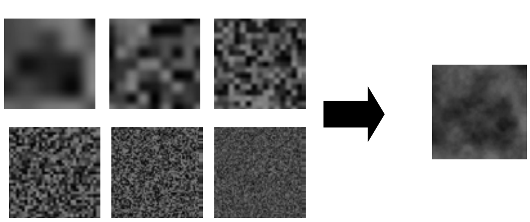


perlin noise – 1D example

step 3: repeat with various amplitudes/frequencies

step 4: add together

perlin noise – 2D example



for more info see Perlin Noise link on proj2 web site

cheap tricks

- texture mapping
- procedural texture mapping
- **bump mapping**
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows