Topic 11:

Texture Mapping

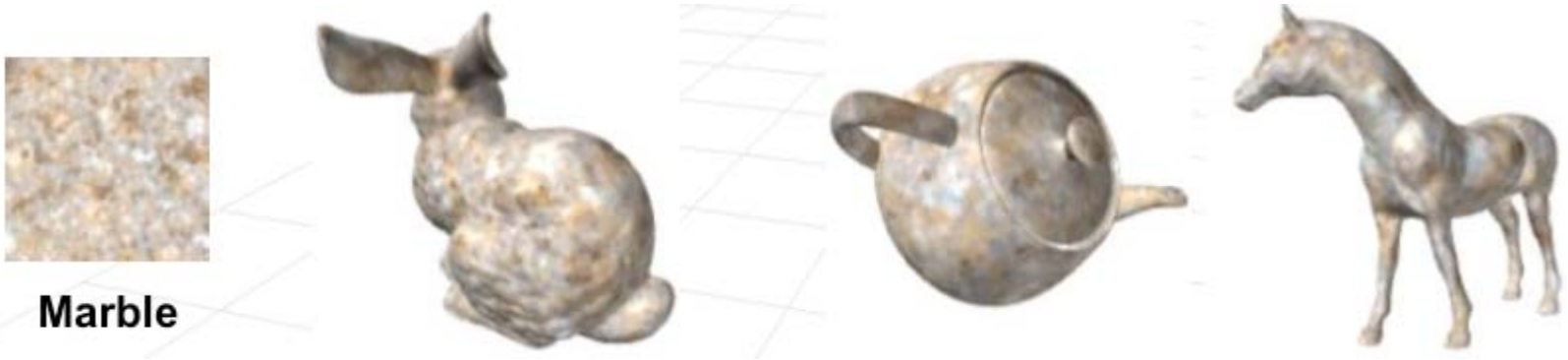
- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping


Guet al, EGSR'07

Rendering w/ no subsurface scattering (opaque skin)

Jensen et al, SIGGRAPH'01

Texture Mapping: Motivation

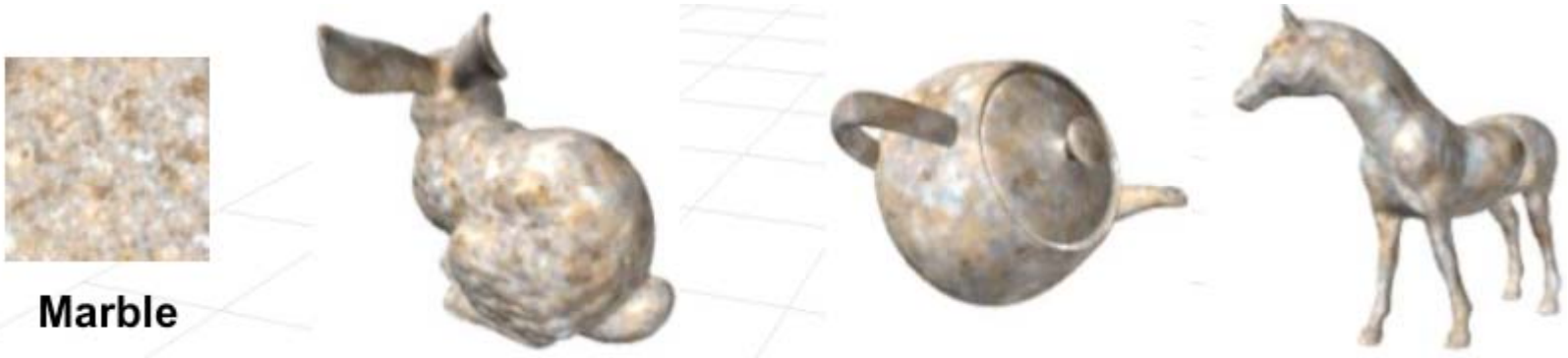


Goal: Endow objects with more varied & realistic appearance through complex variations in reflectance

Key features of texture mapping

- efficient to render
- reusable
- can modulate albedo and/or fine geometry

Introduction to Texture Mapping



Basic questions:

1. Where do textures come from?
2. How do we map textures onto surfaces?
3. How can textures be used to control appearance?
4. How do we integrate texture mapping and scan conversion?

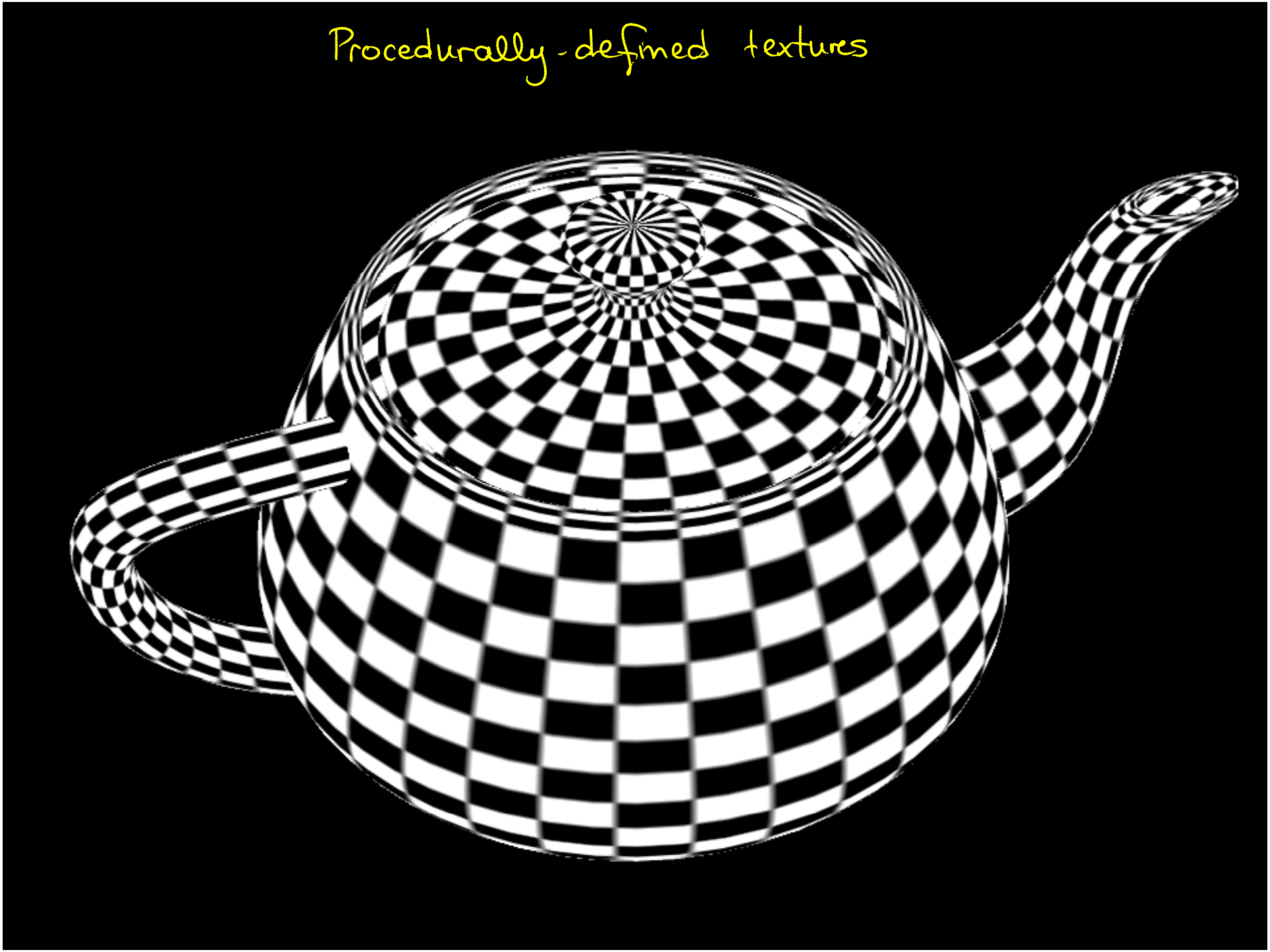
Topic 11:

Texture Mapping

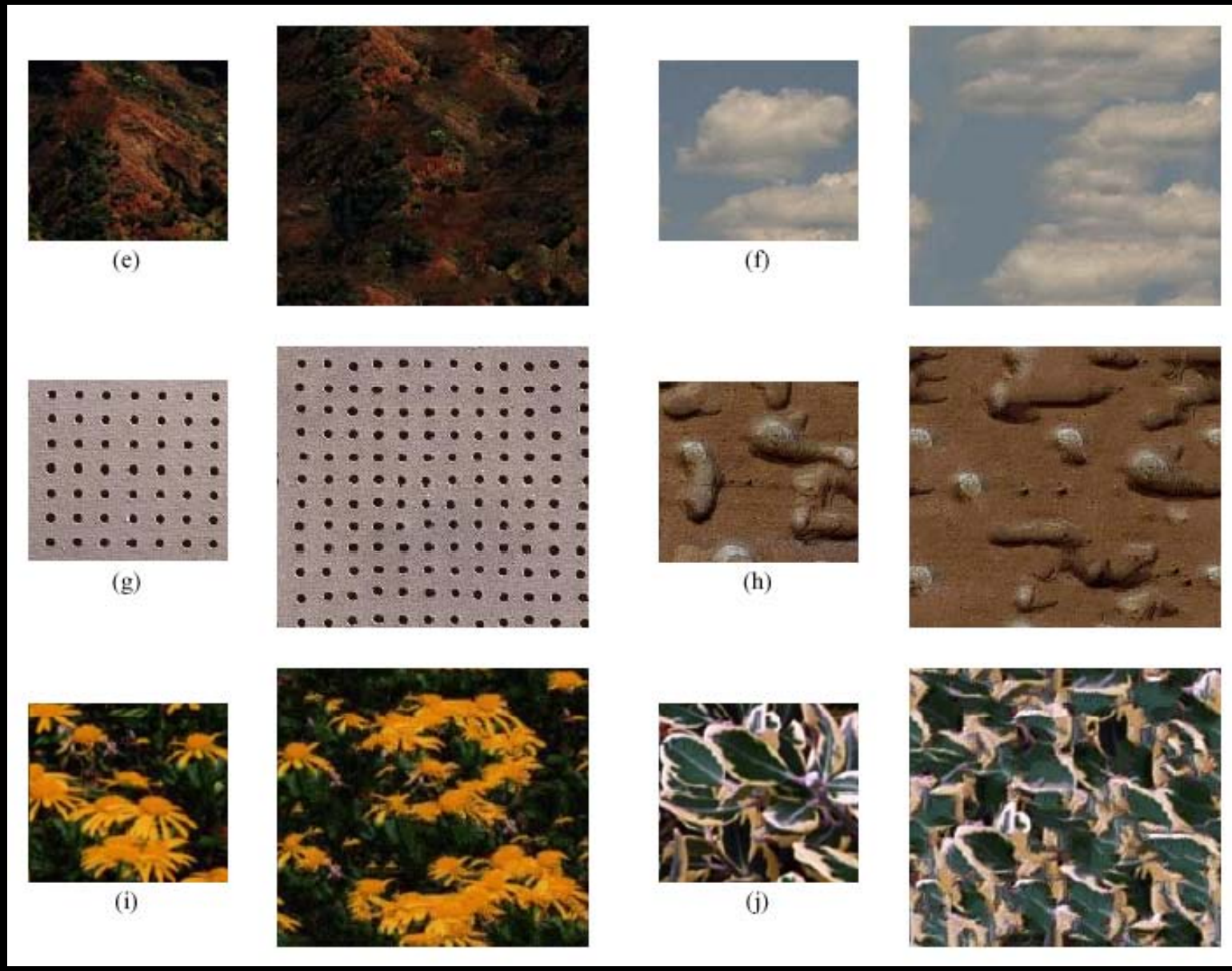
- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

Photos of real materials

Dana et al, TOG-99


Procedurally-defined textures

Texture Synthesis



Kwatra et al, SIGGRAPH '05

Texture Synthesis

Original

Synthesized

Original

Synthesized

Kwatra et al, SIGGRAPH '05

Texture Synthesis

Original

Synthesized

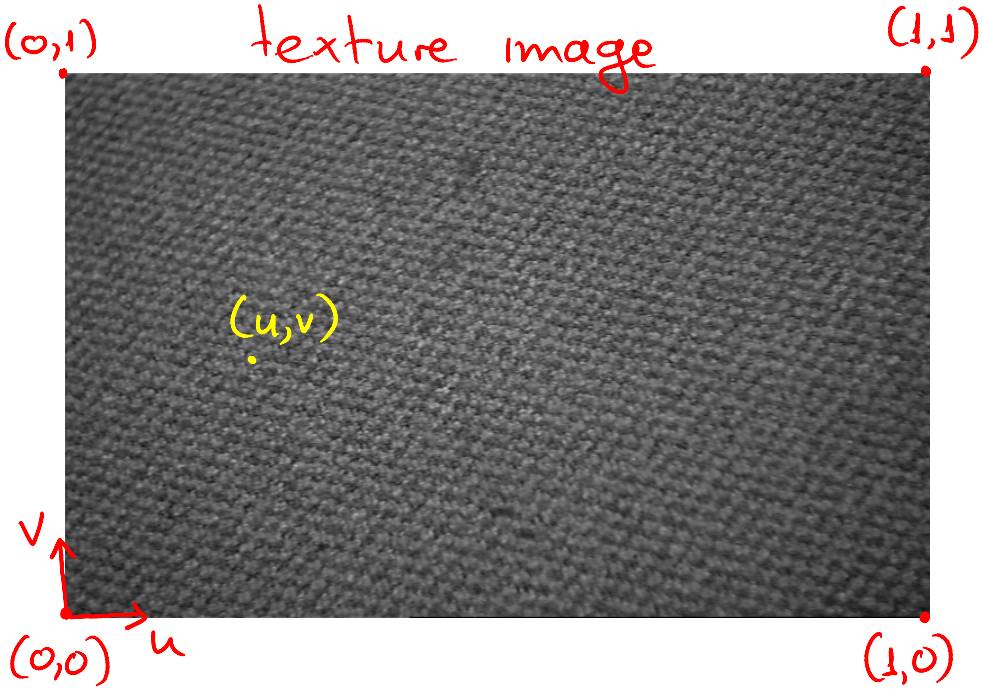
Kwatra et al, SIGGRAPH '05

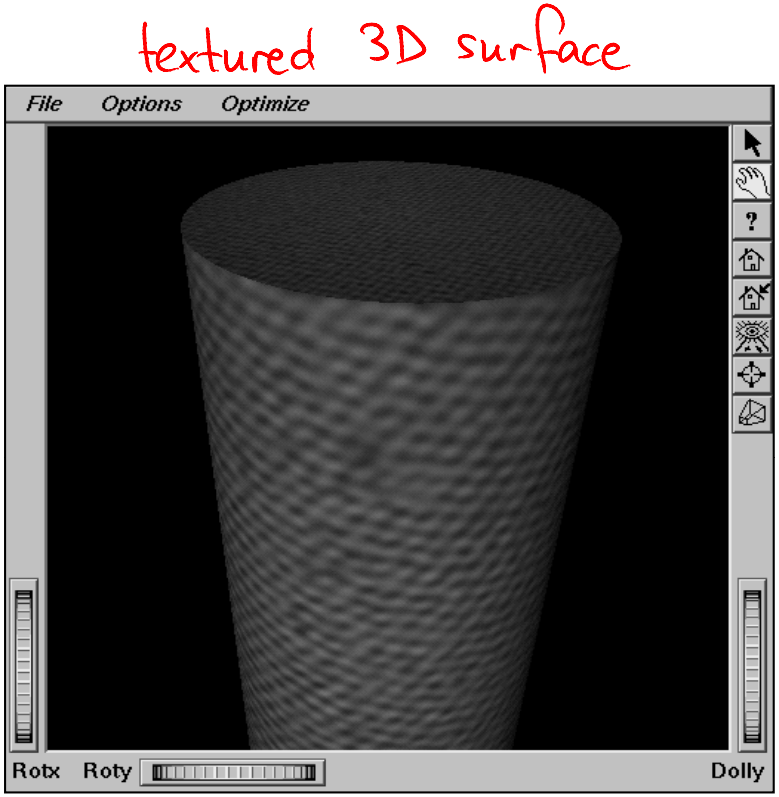
Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

Texture Coordinates

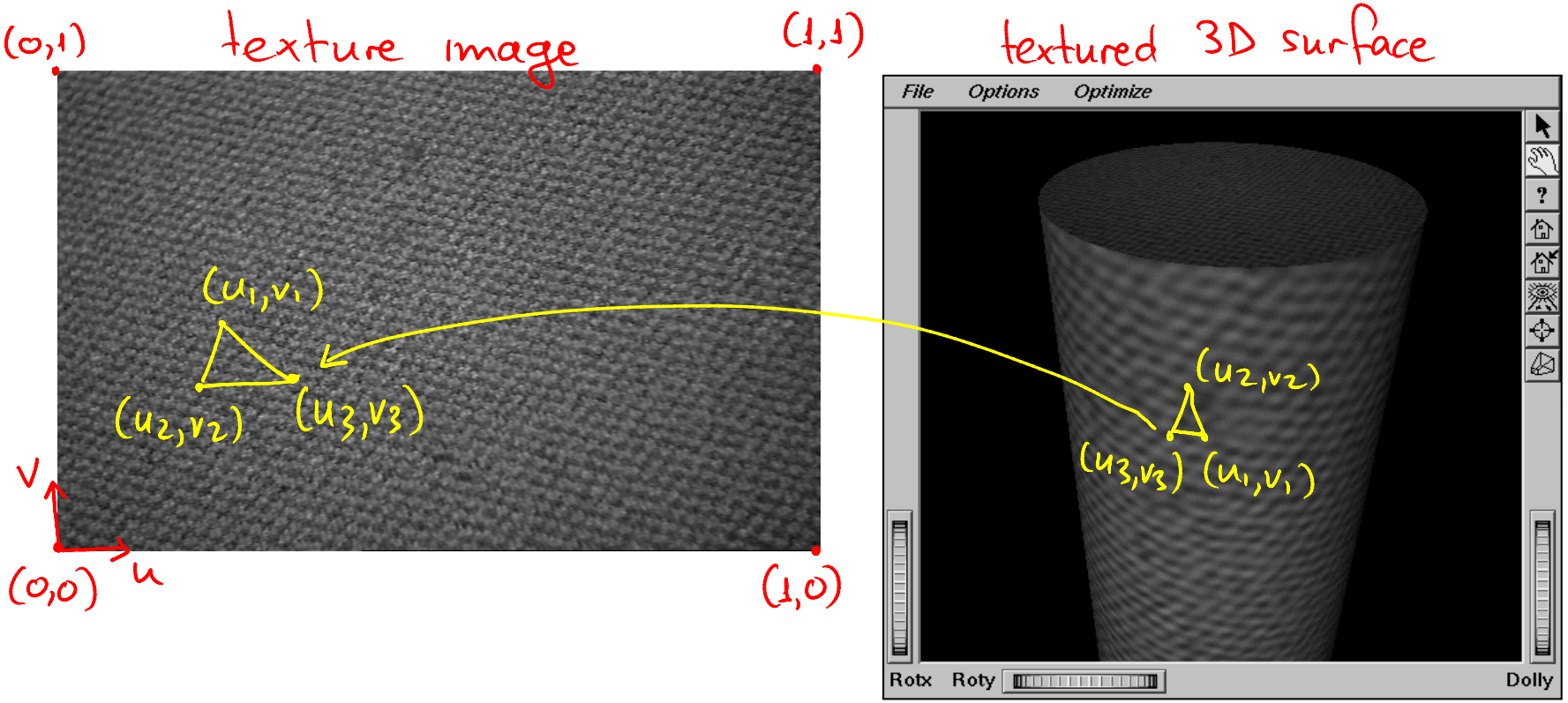




Key ideas

- define a 2D coordinate system for texture image (called texture coordinates)
- define a mapping from triangles to texture coords

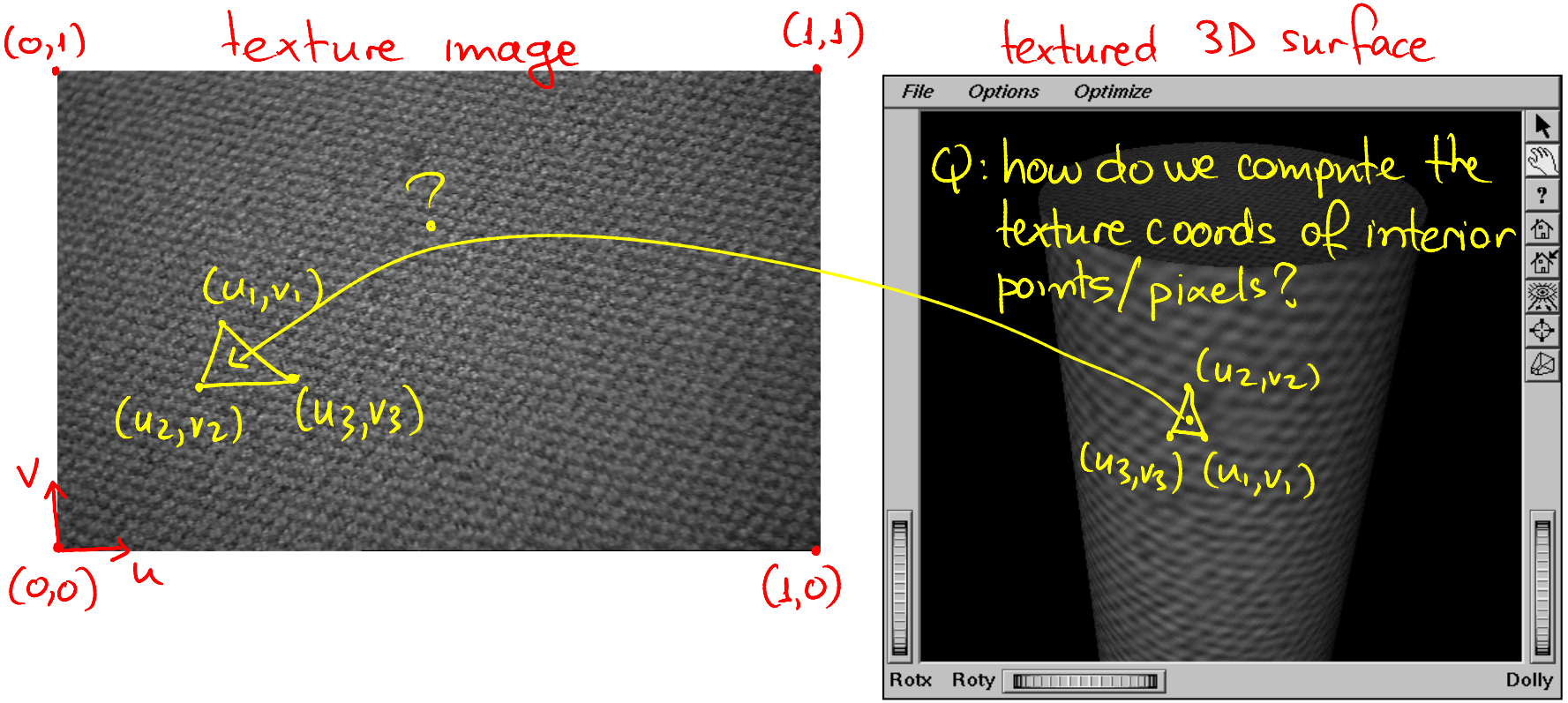
Specifying a Triangle's Texture Coordinates



Two ways to define the mapping:

1. for each face of mesh, specify the texture coordinates of each vertex of the face
2. define a continuous mapping from surface pts to texture pts

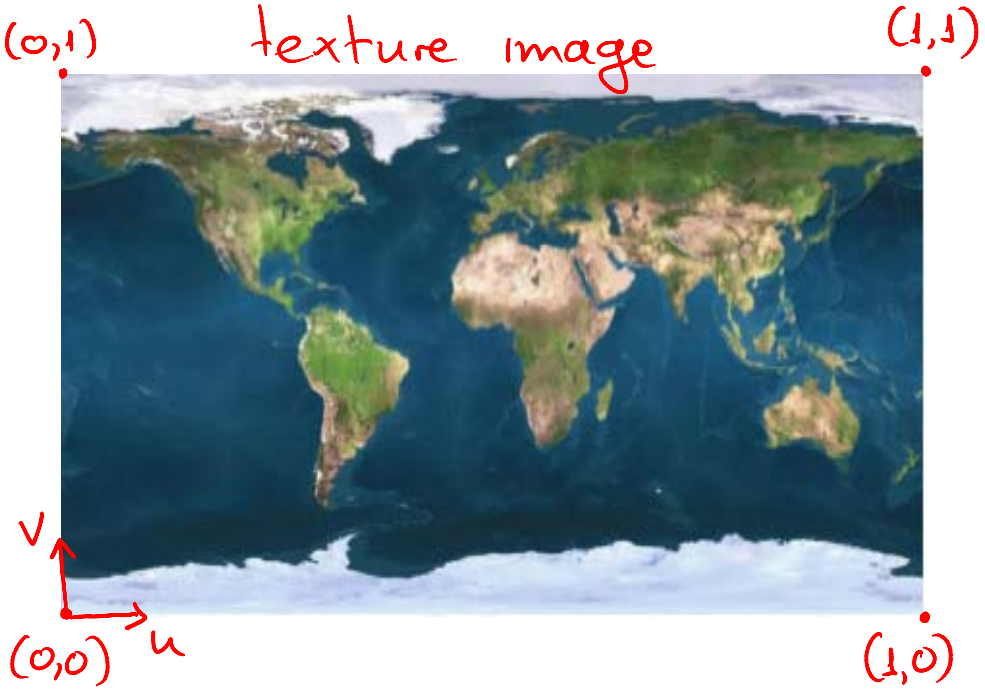
Texture Coords from Triangle Vertices



Two ways to define the mapping:

1. for each face of mesh, specify the texture coordinates of each vertex of the face
2. define a continuous mapping from surface pts to texture pts

Texture Coords from Surface Parameters

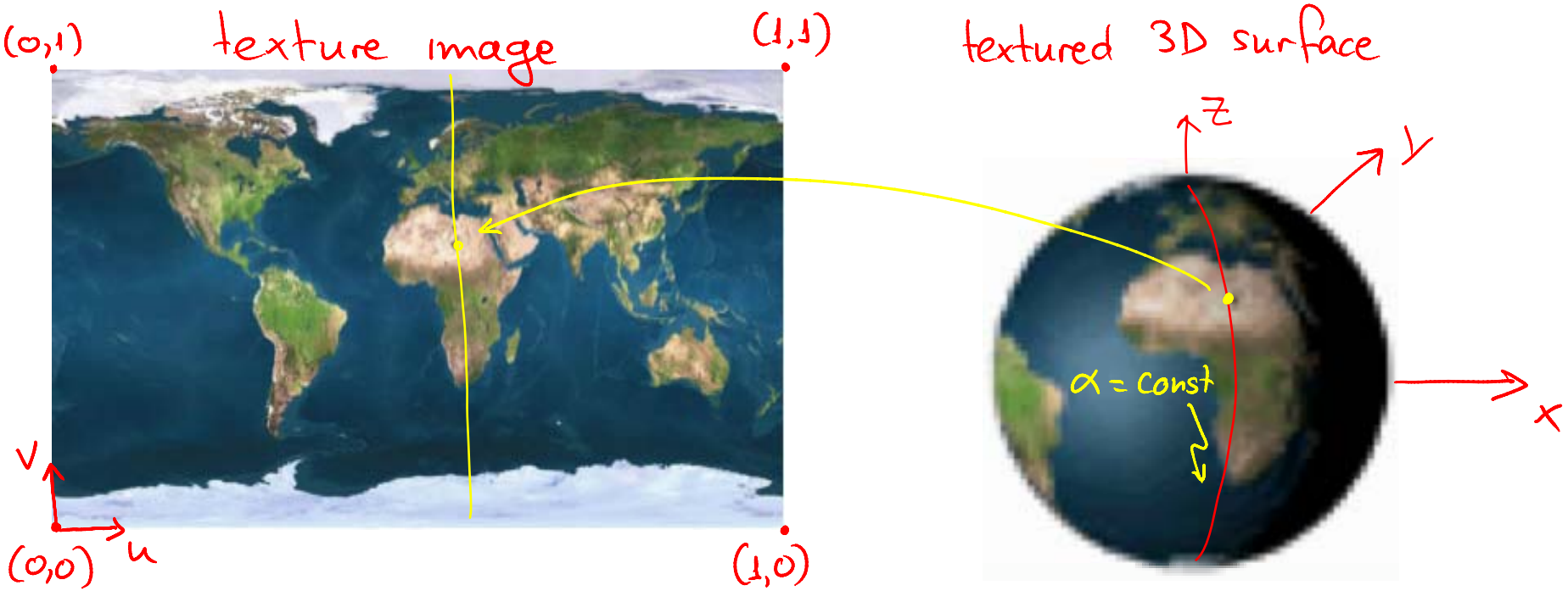


Two ways to define the mapping:

① for each face of mesh, specify the texture coordinates of each vertex of the face

② define a continuous mapping from surface pts to texture pts

Example: Mapping Textures on a Sphere



texture coords of $s(\alpha, \beta)$:

$$u = \frac{\alpha}{2\pi}$$

$$v = \frac{\beta}{\pi}$$

$$\Leftarrow$$

$$s(\alpha, \beta) = \begin{bmatrix} x_0 + r\cos\alpha \sin\beta \\ y_0 + r\sin\alpha \sin\beta \\ z_0 + r\cos\beta \end{bmatrix}$$

$$\alpha \in [0, 2\pi)$$

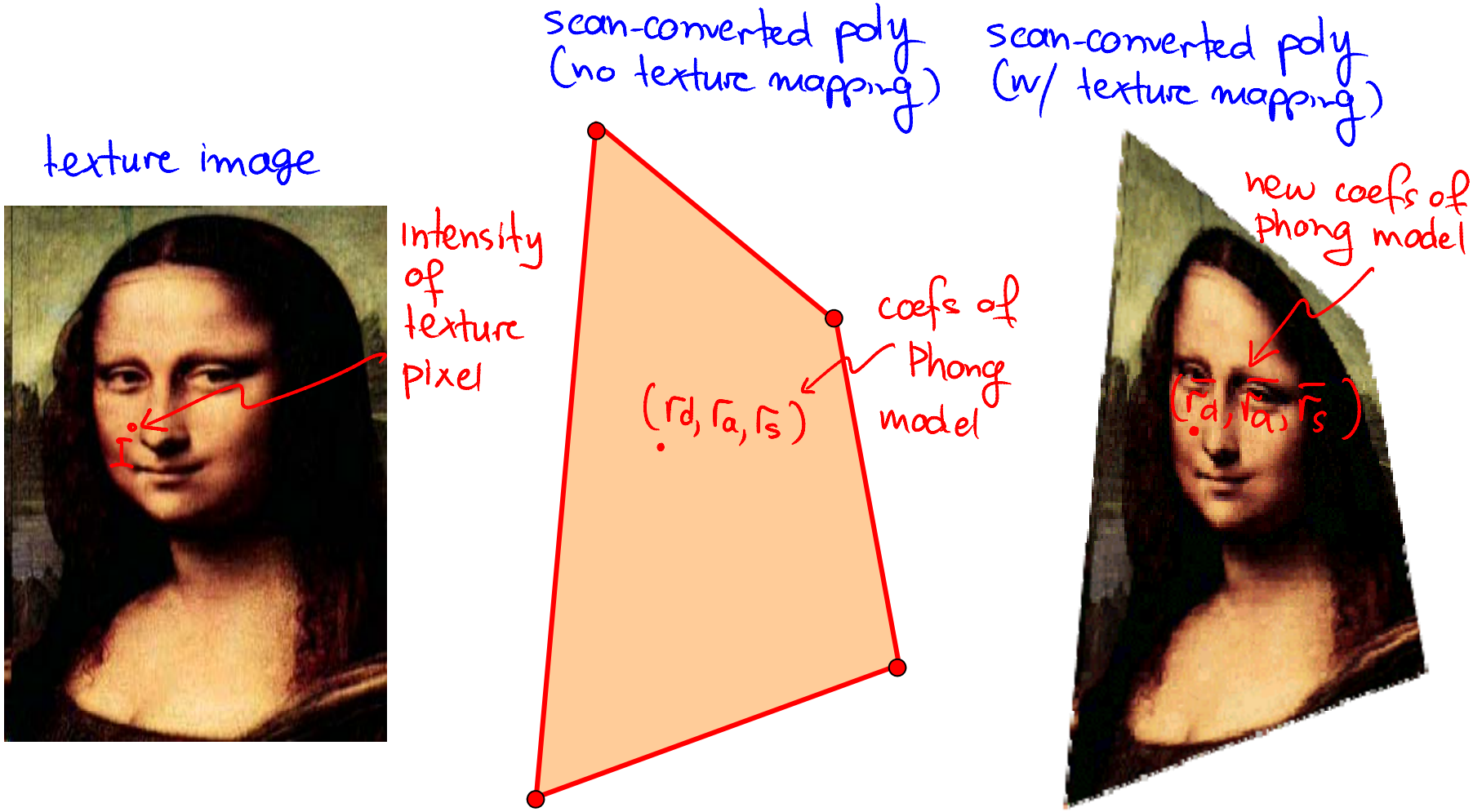
$$\beta \in [0, \pi)$$

Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

Appearance Control via Texture Mapping

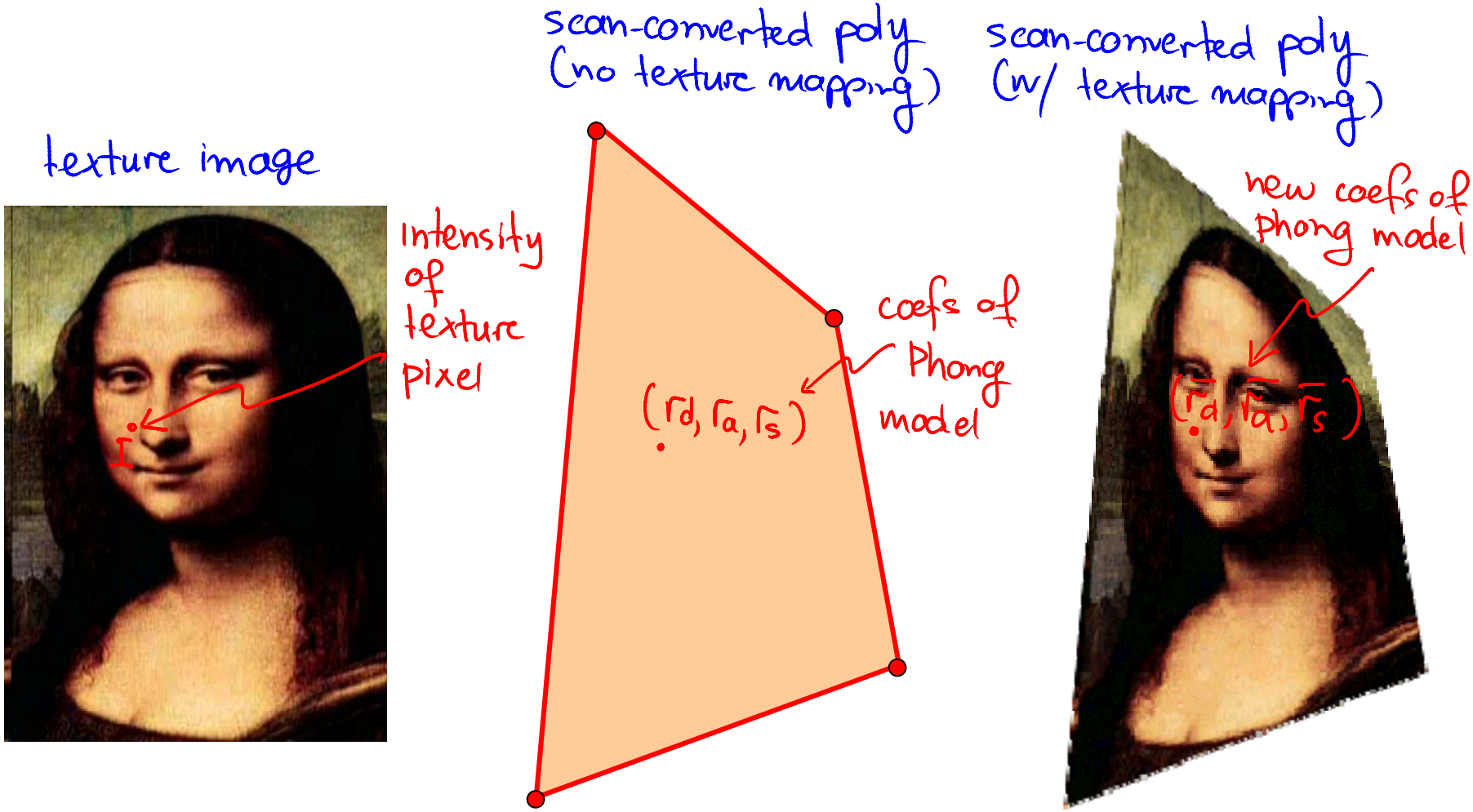


Approach #1 (modulation)

$\bar{r}_d = I \cdot r_d$, $\bar{r}_a = I \cdot r_a$, $\bar{r}_s = I \cdot r_s$

(assumes I normalized to range [0,1])

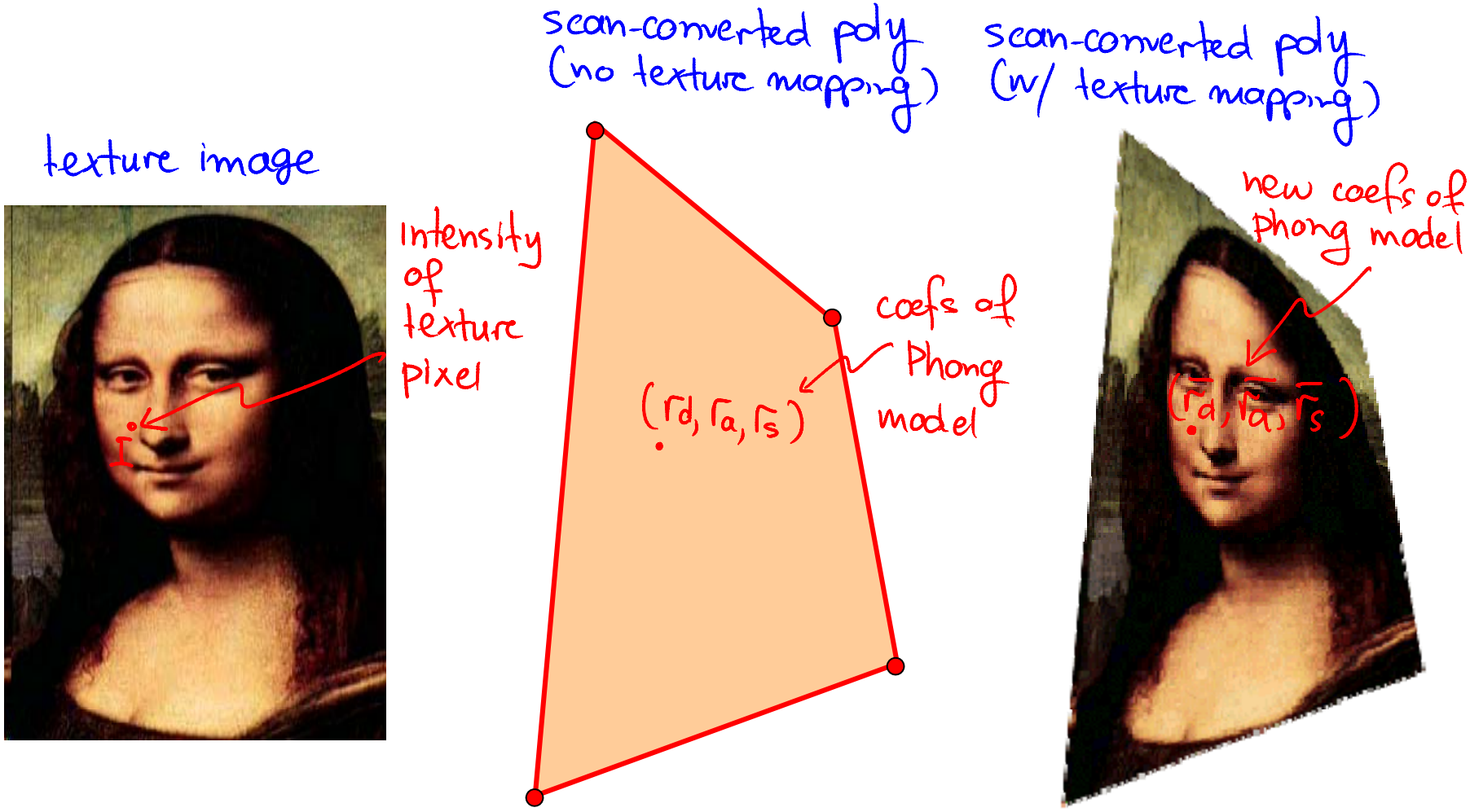
Appearance Control via Texture Mapping



Approach #2 (replacement)

$\bar{r}_d = I$ $\bar{r}_a = I$ $\bar{r}_s = I$

Appearance Control via Texture Mapping



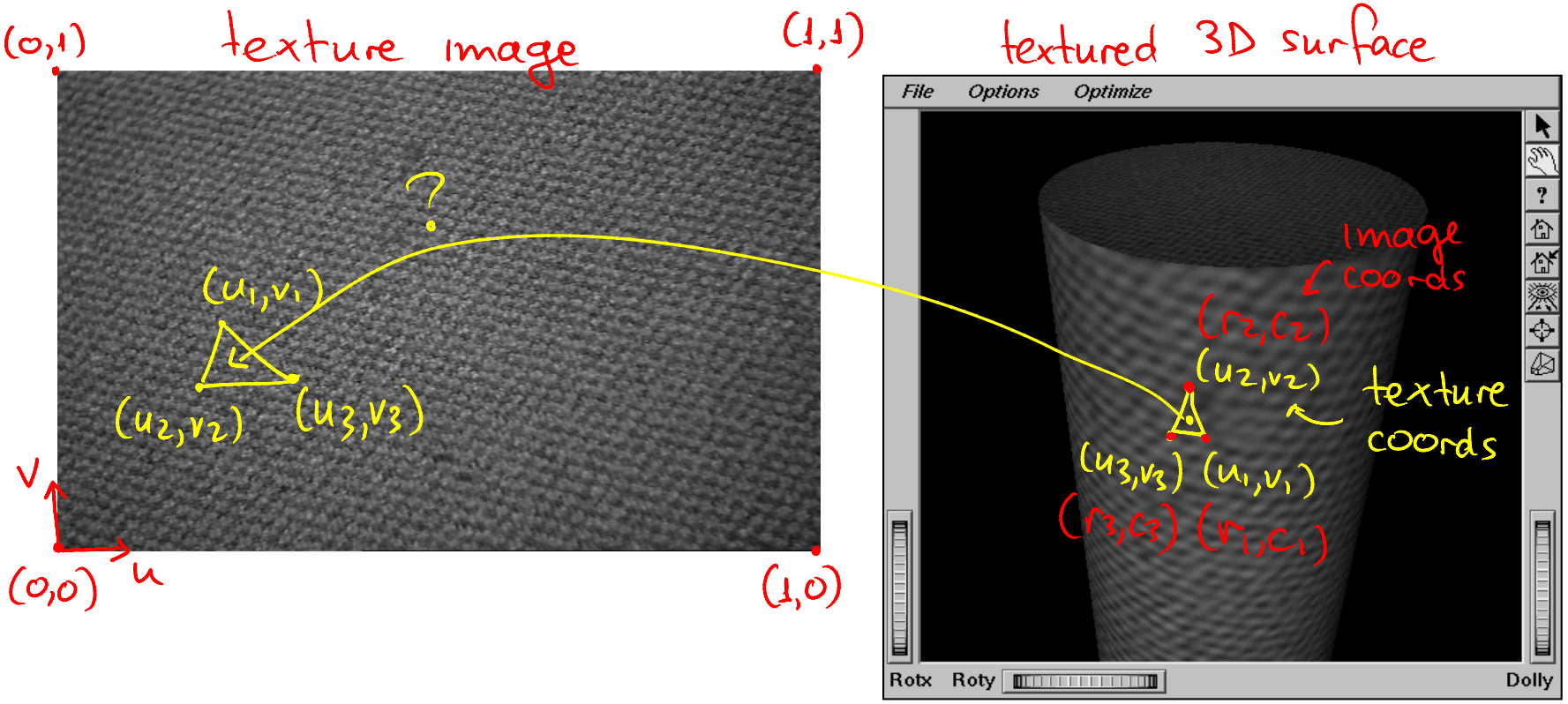
Other options also possible – see glTexEnvf()
in OpenGL

Topic 11:

Texture Mapping

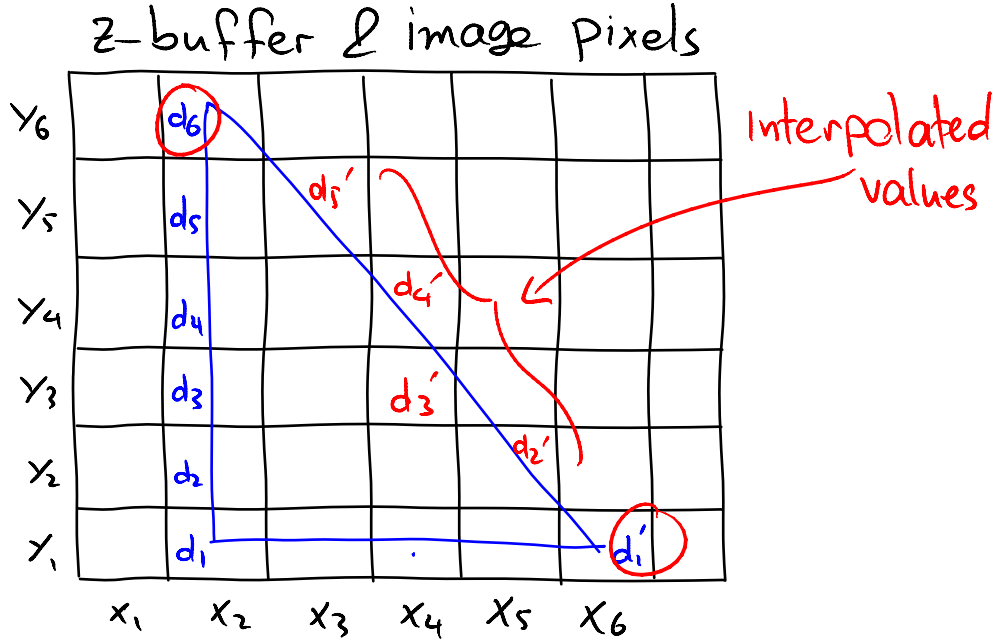
- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

Texture Coords of Pixels in Triangle's Interior



Q: how do we determine the texture coordinates of interior polygon pixels during scan-conversion?

Reminder: Basic Scan Conversion



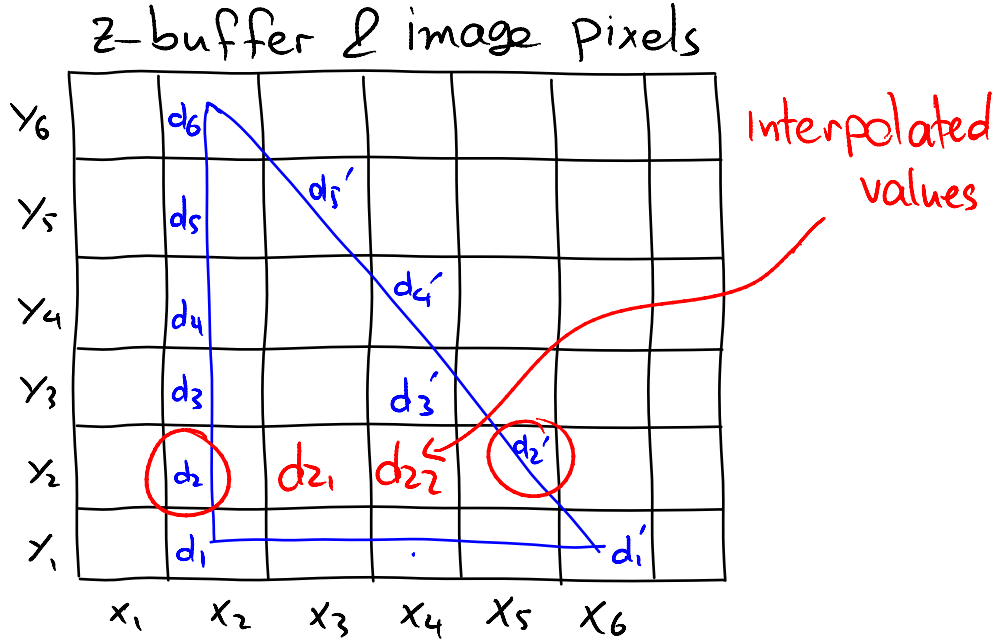
Step a:

for each edge,
interpolate d, L, ...
between its two
vertices

step b:

for each scanline
interpolate d, L, ...
between two edge
pixels

Reminder: Basic Scan Conversion



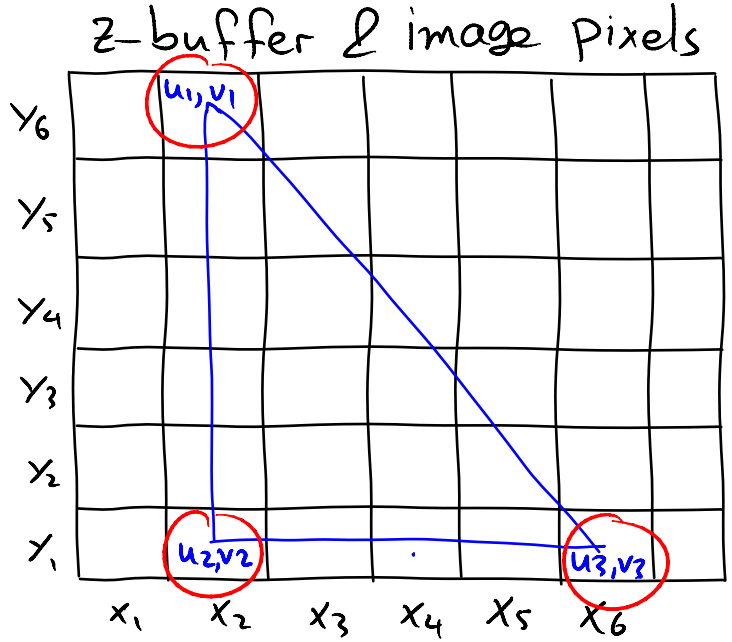
Step a:

for each edge,
interpolate d, L, ...
between its two
vertices

Step b:

for each scanline
interpolate d, L, ...
between two edge
pixels

Should we Interpolate Texture Coordinates?



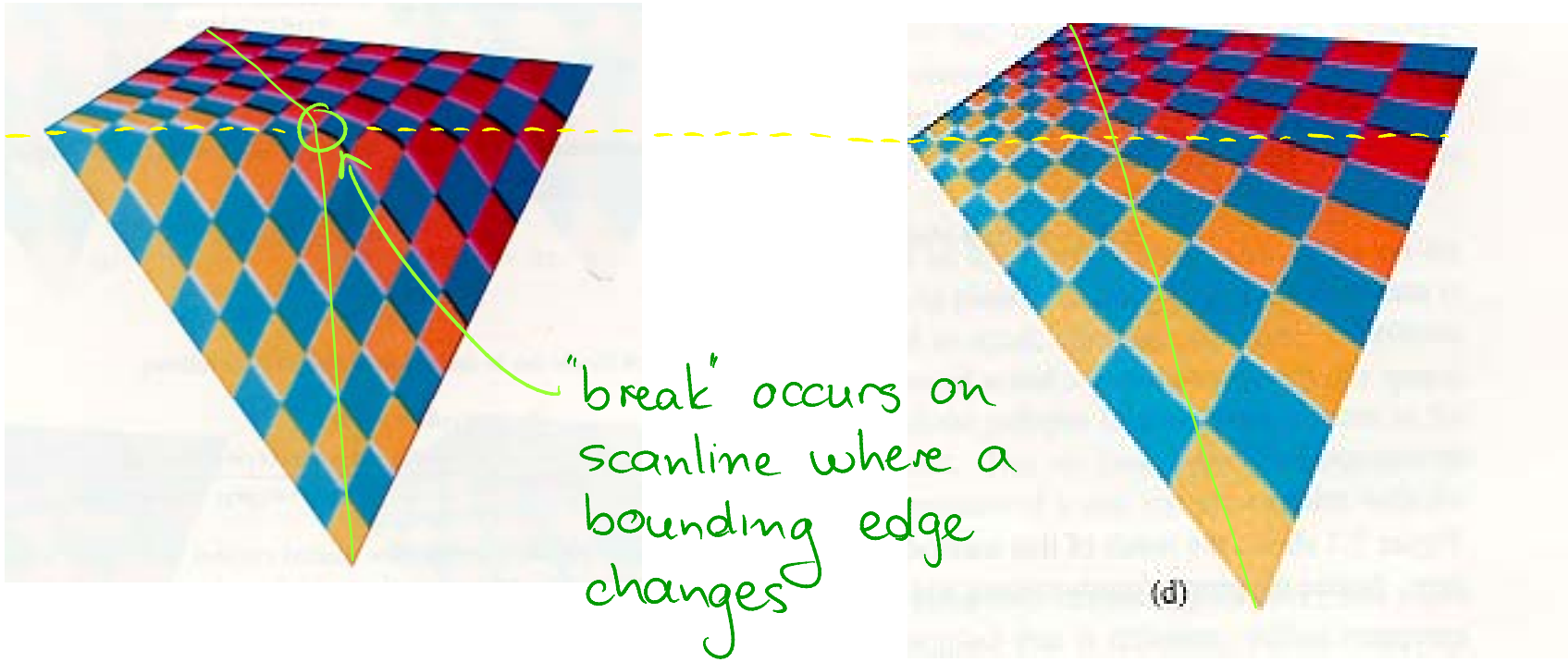
Q: Will this interpolation scheme also work for texture coordinates?

Not very well!

Artifacts Caused by Naive Scan Conversion

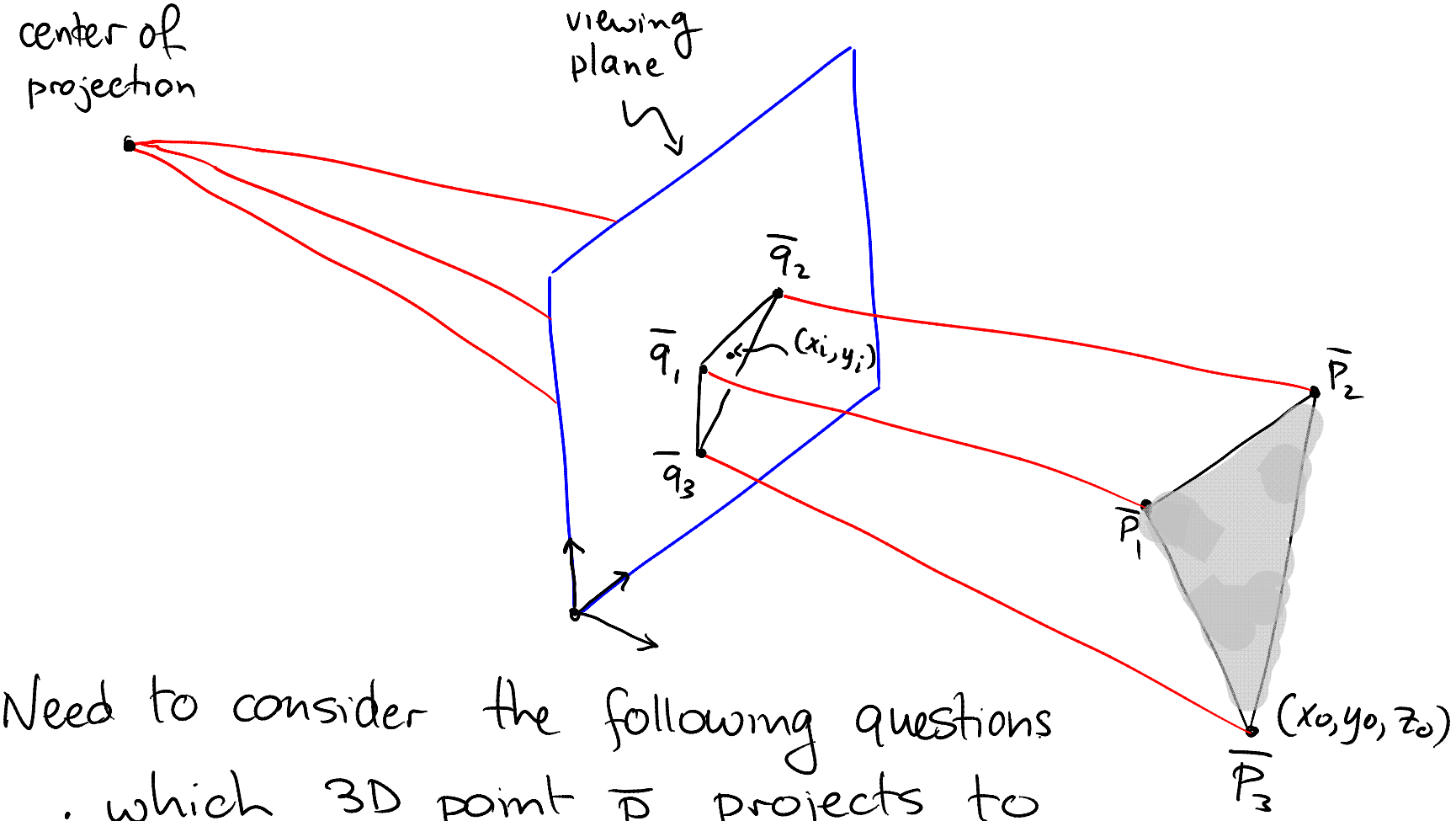
Scan-converted polygon

Correct appearance



Wolberg, 1991

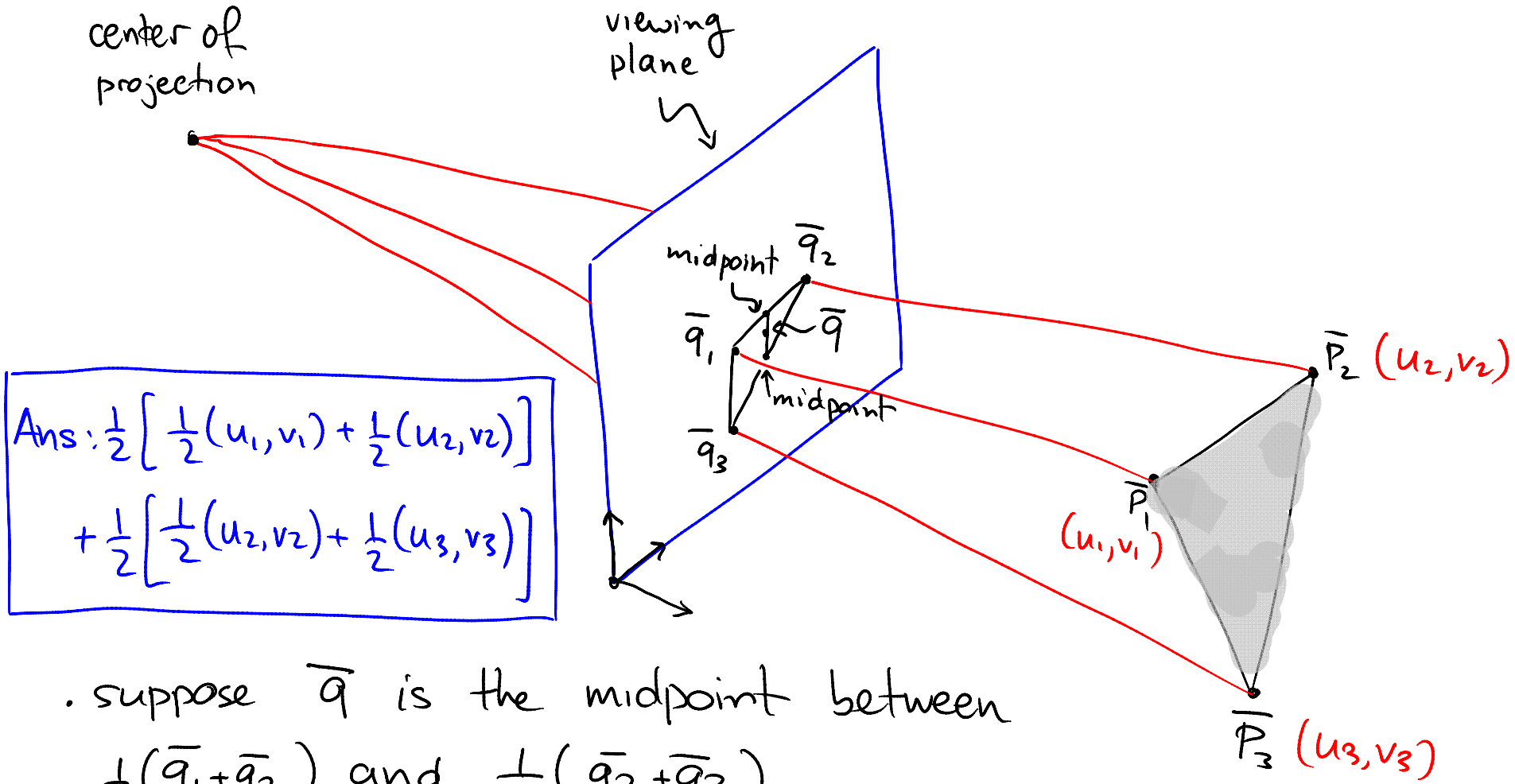
Why Do These Artifacts Occur?



Need to consider the following questions

- which 3D point $\bar{P}$ projects to pixel $\bar{q} = (x_i, y_i)$?
- what are the texture coords of point $\bar{P}$?

Why Do These Artifacts Occur?

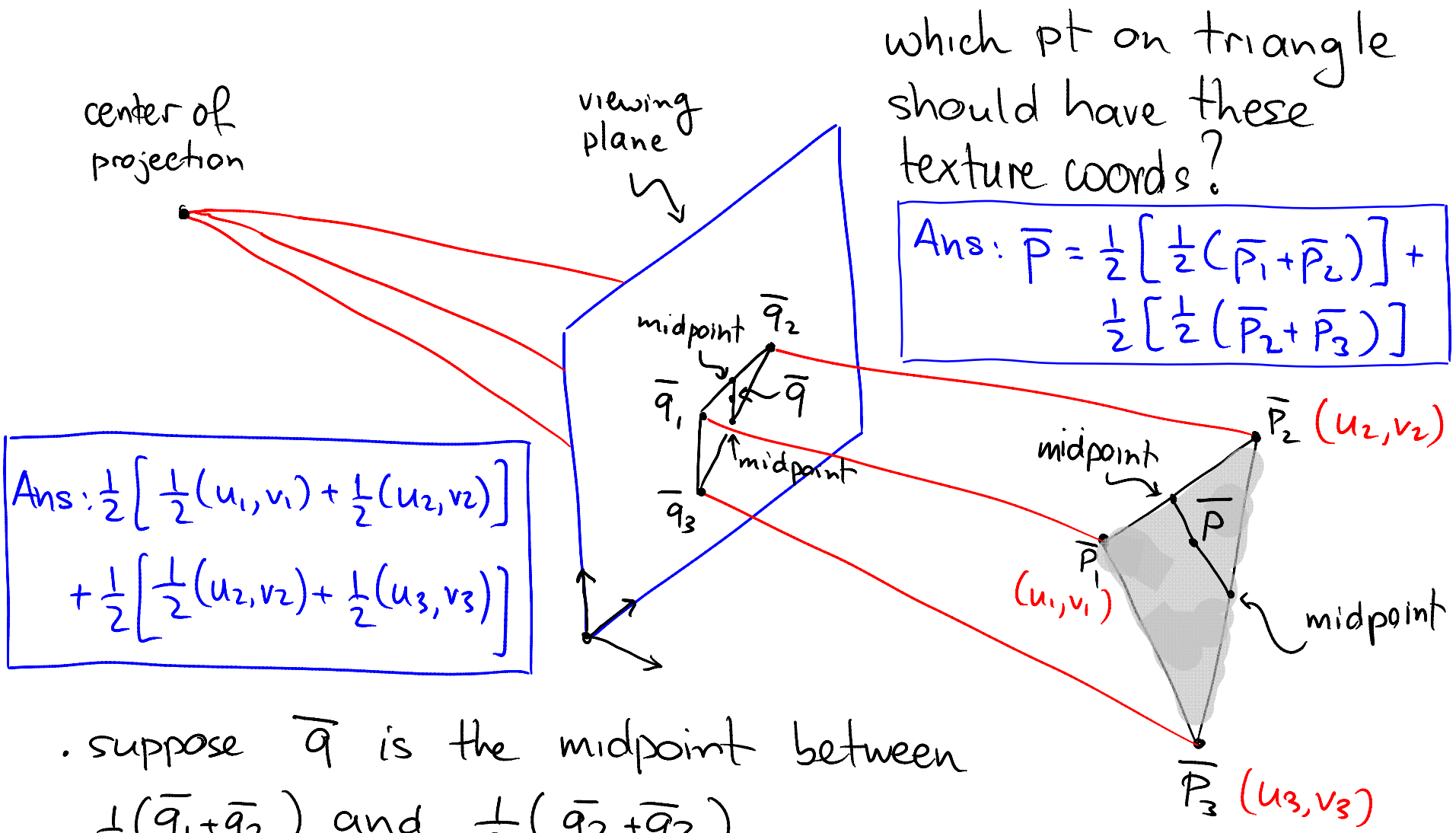


Ans: $\frac{1}{2}\left[\frac{1}{2}(u_1, v_1) + \frac{1}{2}(u_2, v_2)\right]$

$+ \frac{1}{2}\left[\frac{1}{2}(u_2, v_2) + \frac{1}{2}(u_3, v_3)\right]$

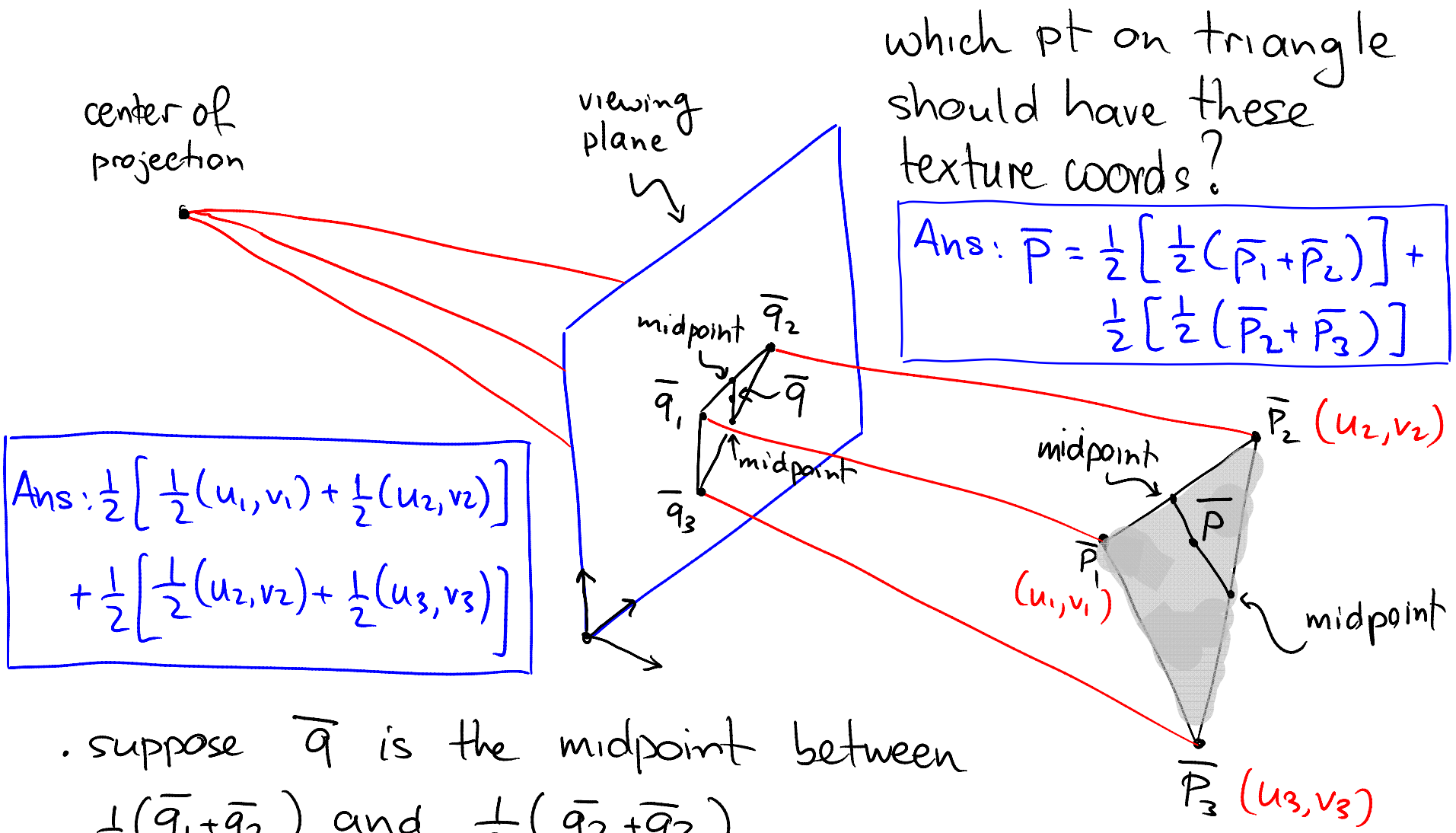
- suppose $\bar{q}$ is the midpoint between $\frac{1}{2}(\bar{q}_1 + \bar{q}_2)$ and $\frac{1}{2}(\bar{q}_2 + \bar{q}_3)$
- what texture coords will be assigned to $\bar{q}$?

Why Do These Artifacts Occur?



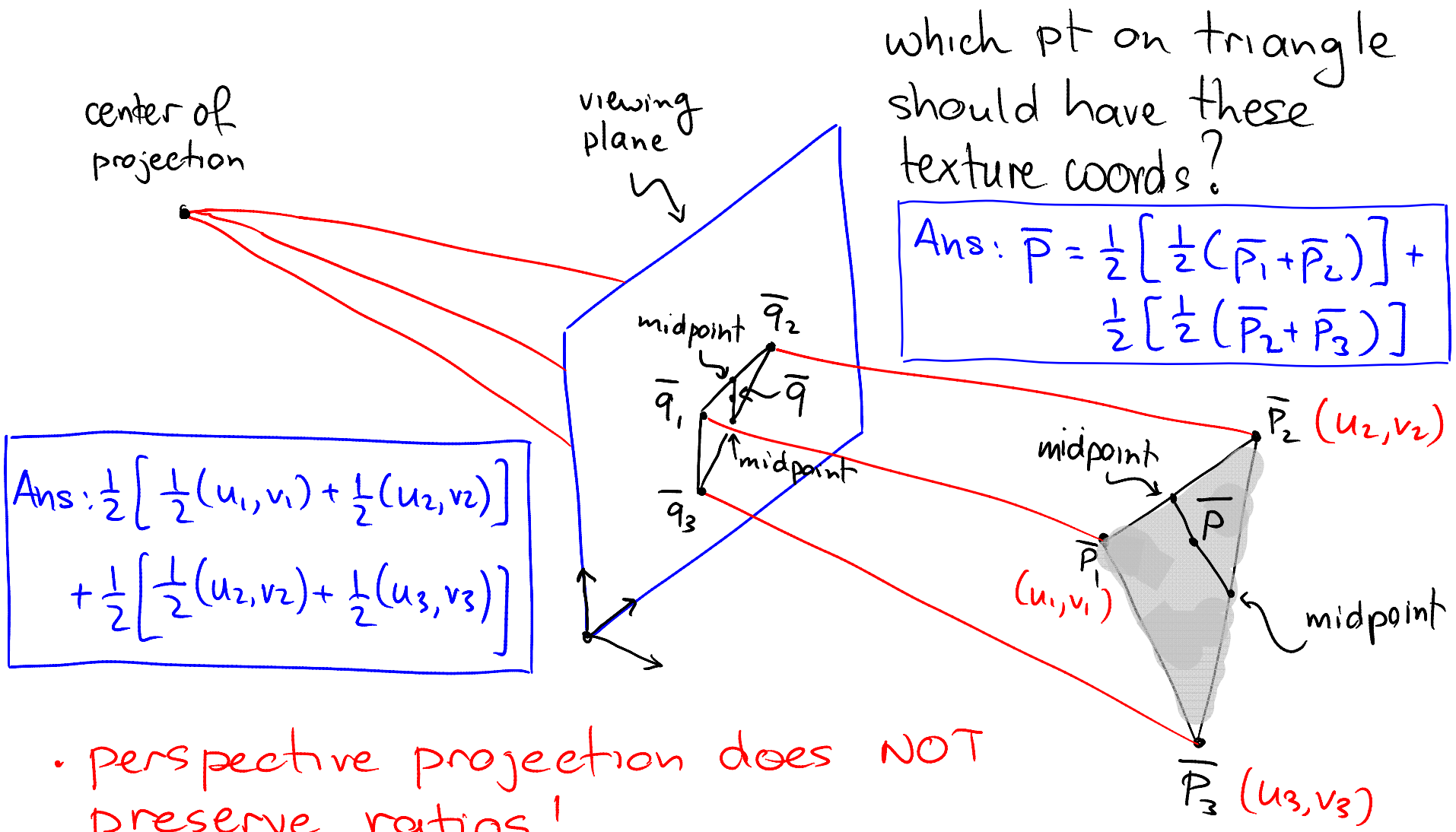
- suppose $\bar{q}$ is the midpoint between $\frac{1}{2}(\bar{q}_1+\bar{q}_2)$ and $\frac{1}{2}(\bar{q}_2+\bar{q}_3)$
- what texture coords will be assigned to $\bar{q}$?

Why Do These Artifacts Occur?



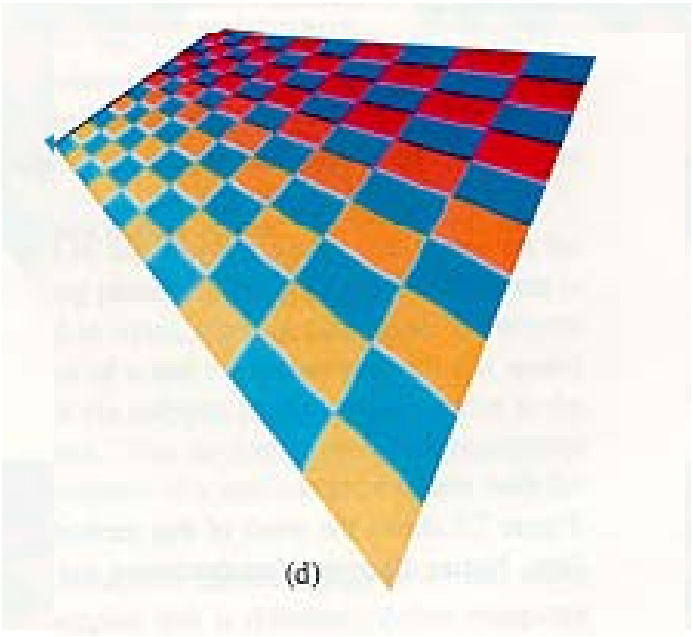
- suppose $\bar{q}$ is the midpoint between $\frac{1}{2}(\bar{q}_1+\bar{q}_2)$ and $\frac{1}{2}(\bar{q}_2+\bar{q}_3)$
- problem: $\bar{p}$ will <u>not</u> project to $\bar{q}$ in general

Why Do These Artifacts Occur?

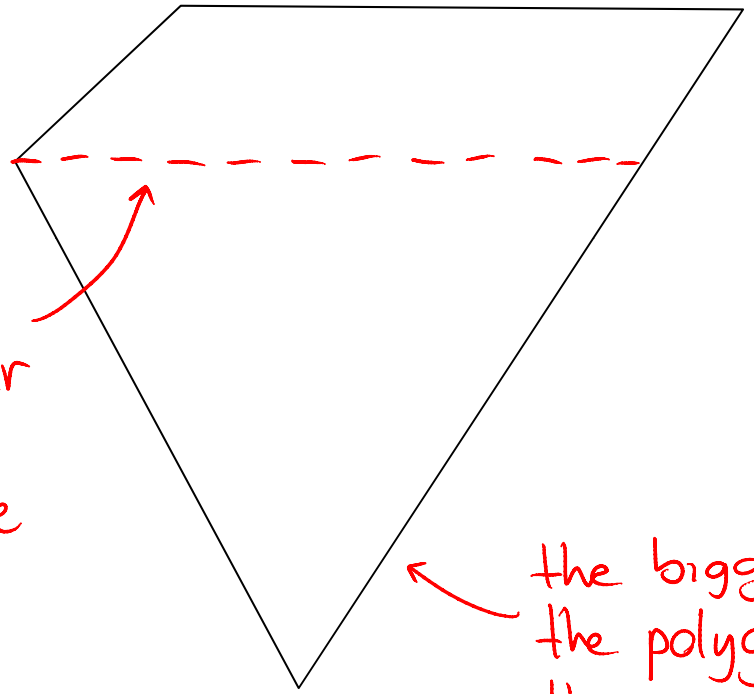


- perspective projection does NOT preserve ratios!
- problem: $\bar{p}$ will <u>not</u> project to $\bar{q}$ in general

Reducing Artifacts of Naive Scan Conversion

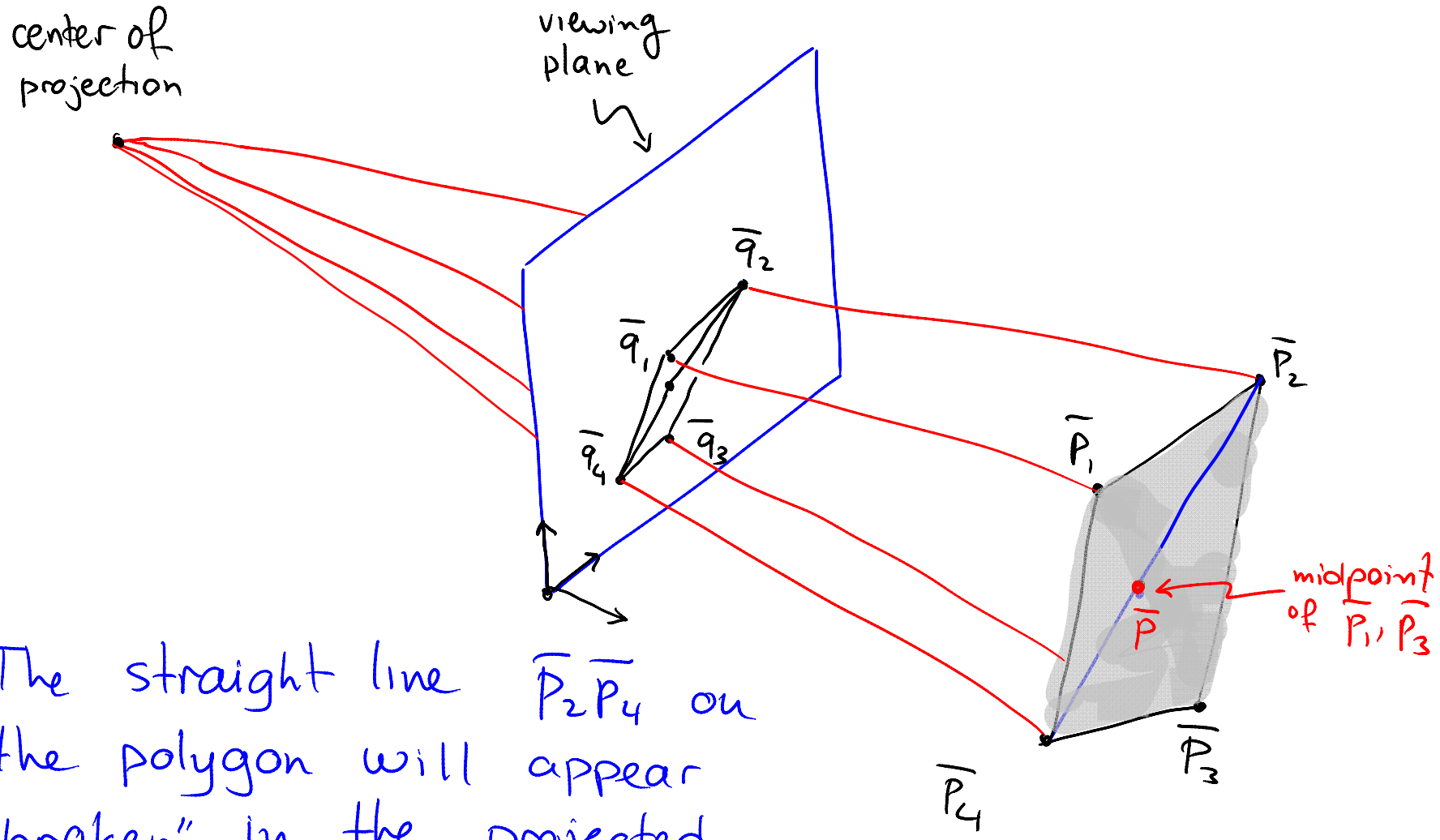


break will occur on this scanline



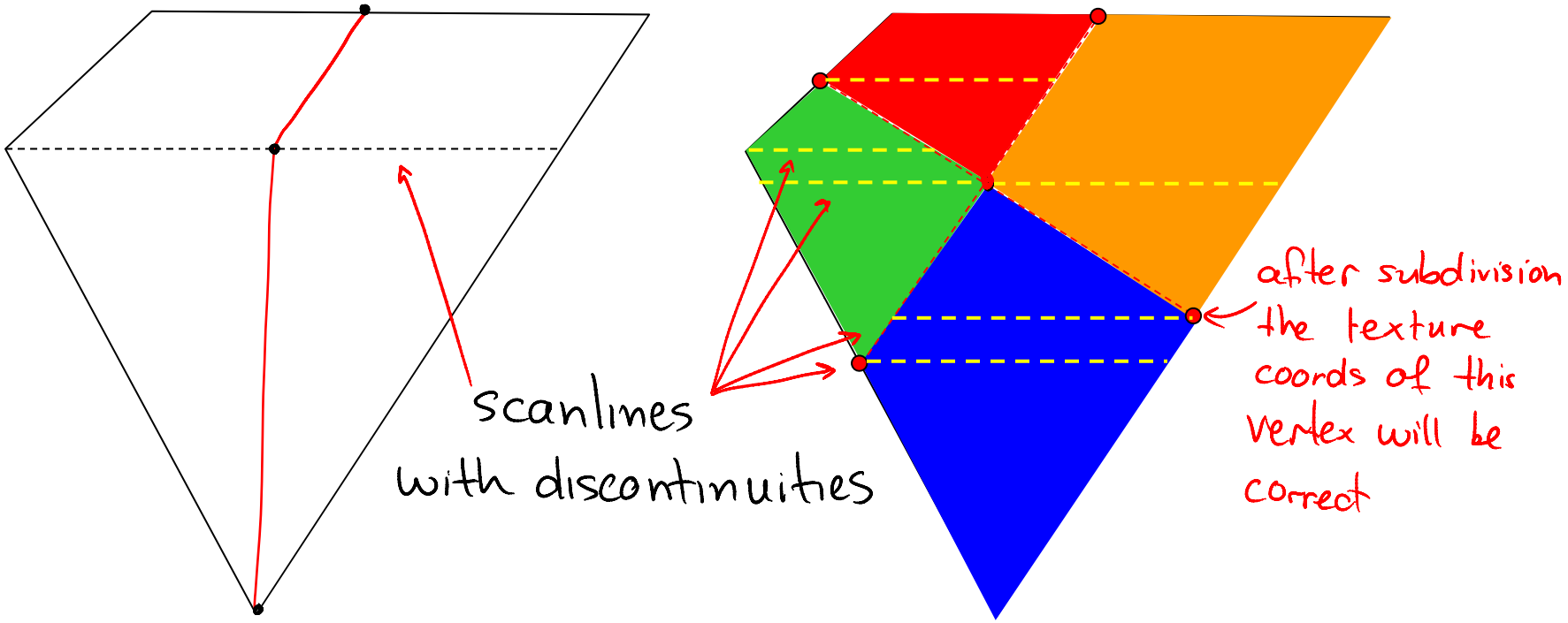
the bigger the polygon, the more noticeable the artifacts

Reducing Artifacts of Naive Scan Conversion



The straight line $\bar{P}_2\bar{P}_4$ on the polygon will appear "broken" in the projected triangle

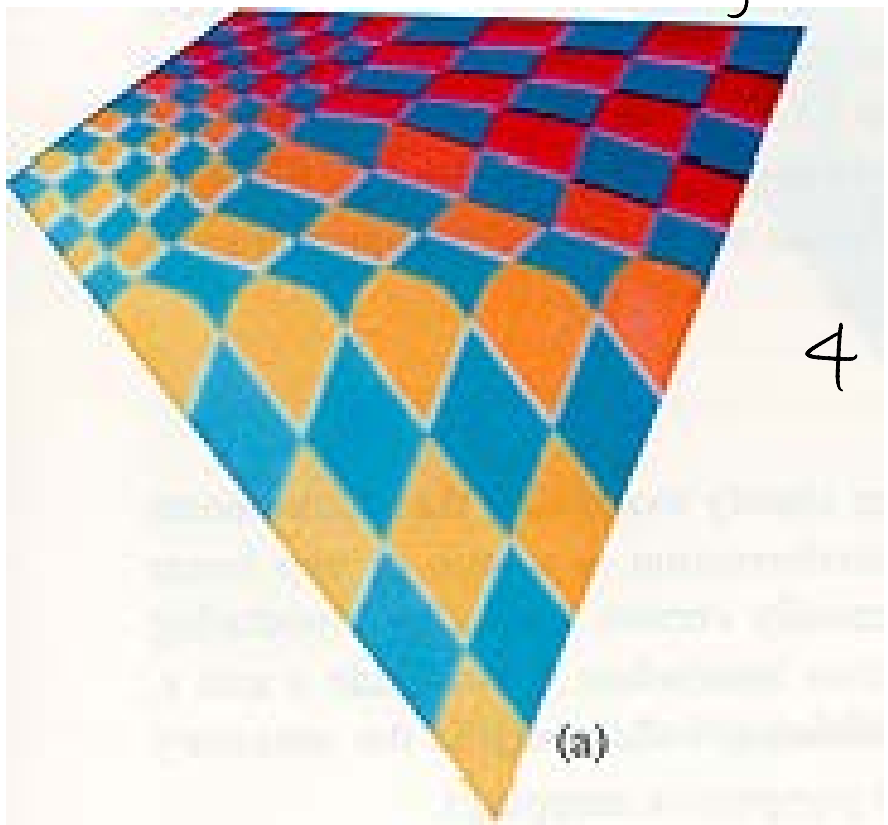
Reducing Artifacts of Naive Scan Conversion



It is possible to reduce the magnitude of distortions by subdividing the polygon

Reducing Artifacts by Polygon Subdivision

Wolberg, 1991

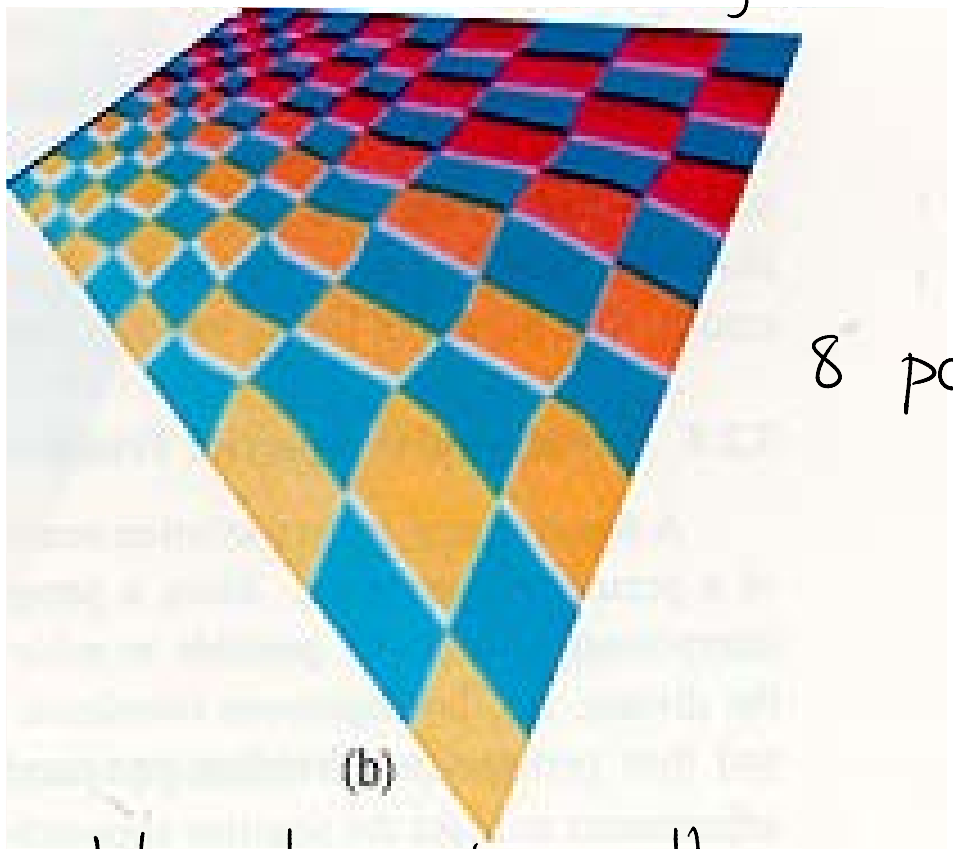


4 polys

It is possible to reduce the magnitude of distortions by subdividing the polygon

Reducing Artifacts by Polygon Subdivision

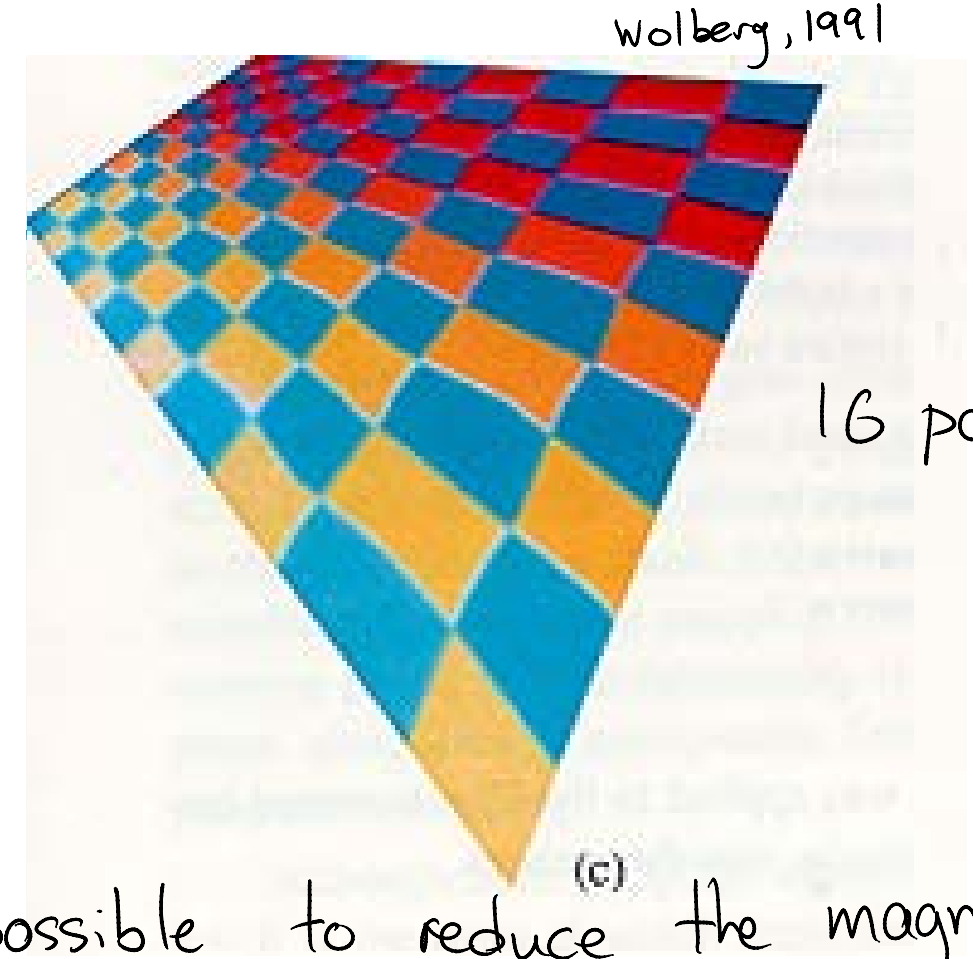
Wolberg, 1991



8 polys

It is possible to reduce the magnitude of distortions by subdividing the polygon

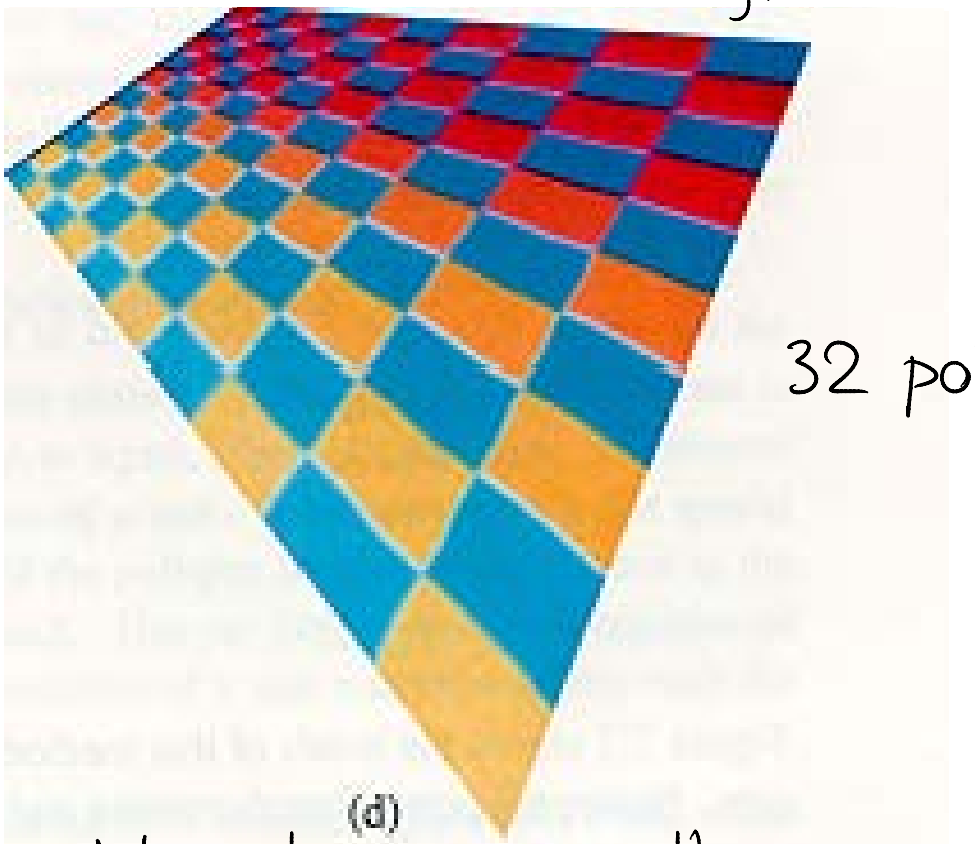
Reducing Artifacts by Polygon Subdivision



It is possible to reduce the magnitude of distortions by subdividing the polygon

Reducing Artifacts by Polygon Subdivision

Wolberg, 1991



32 polys

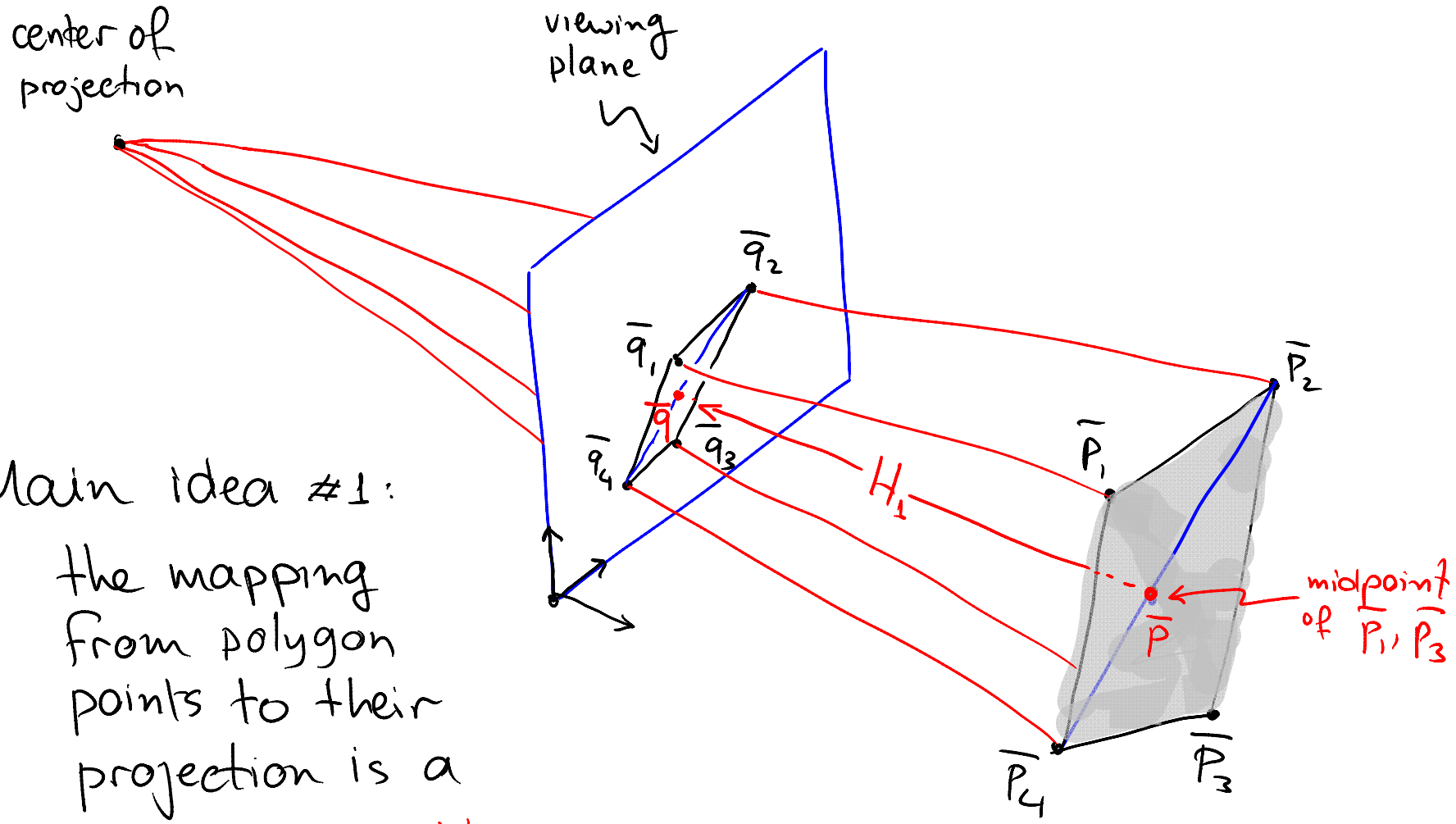
It is possible to reduce the magnitude of distortions by subdividing the polygon

Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

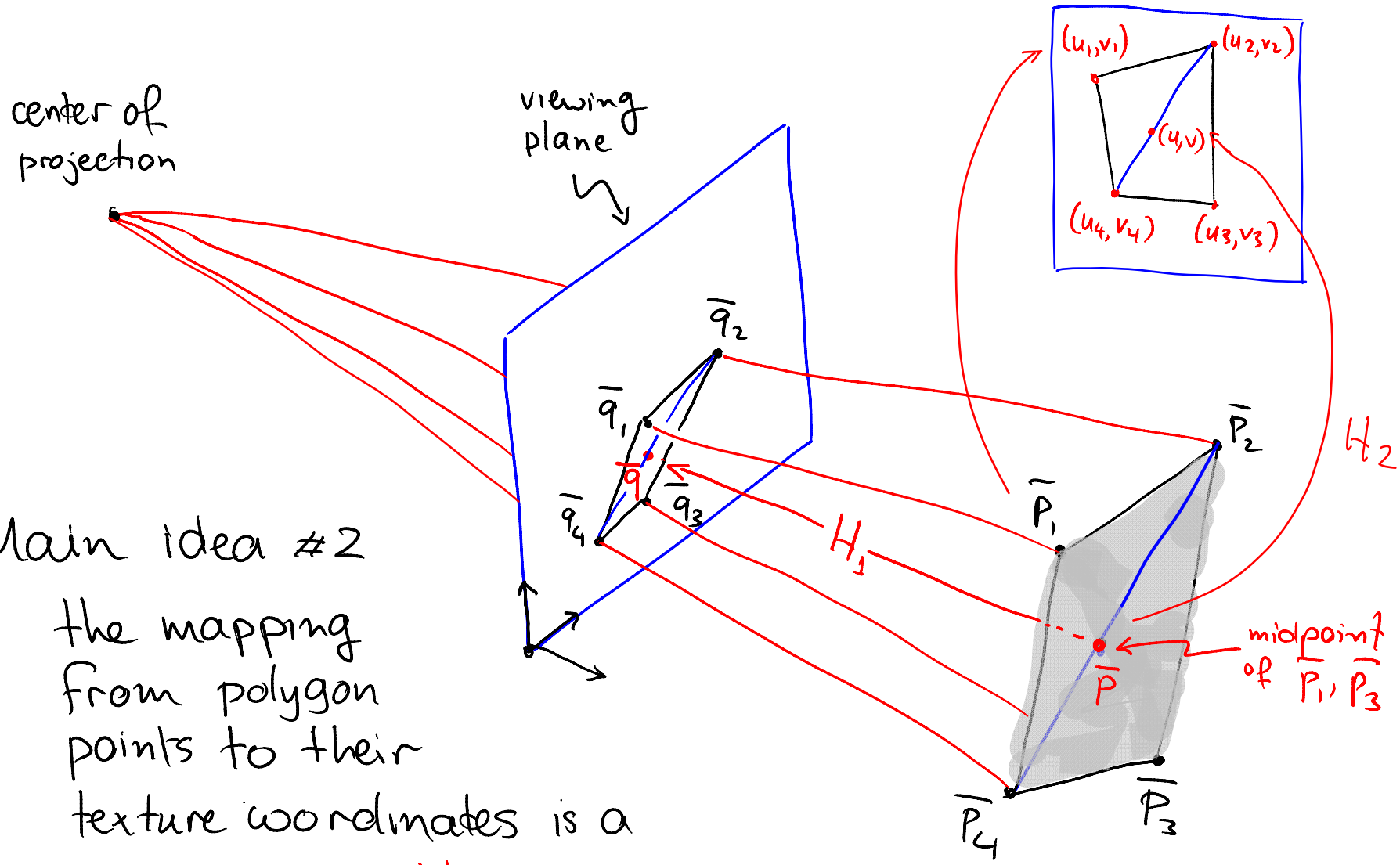
Perspectively-Correct Texture Mapping



Main idea #1:

the mapping from polygon points to their projection is a homography H_1 ↖ most general transform that maps 2D lines to 2D lines

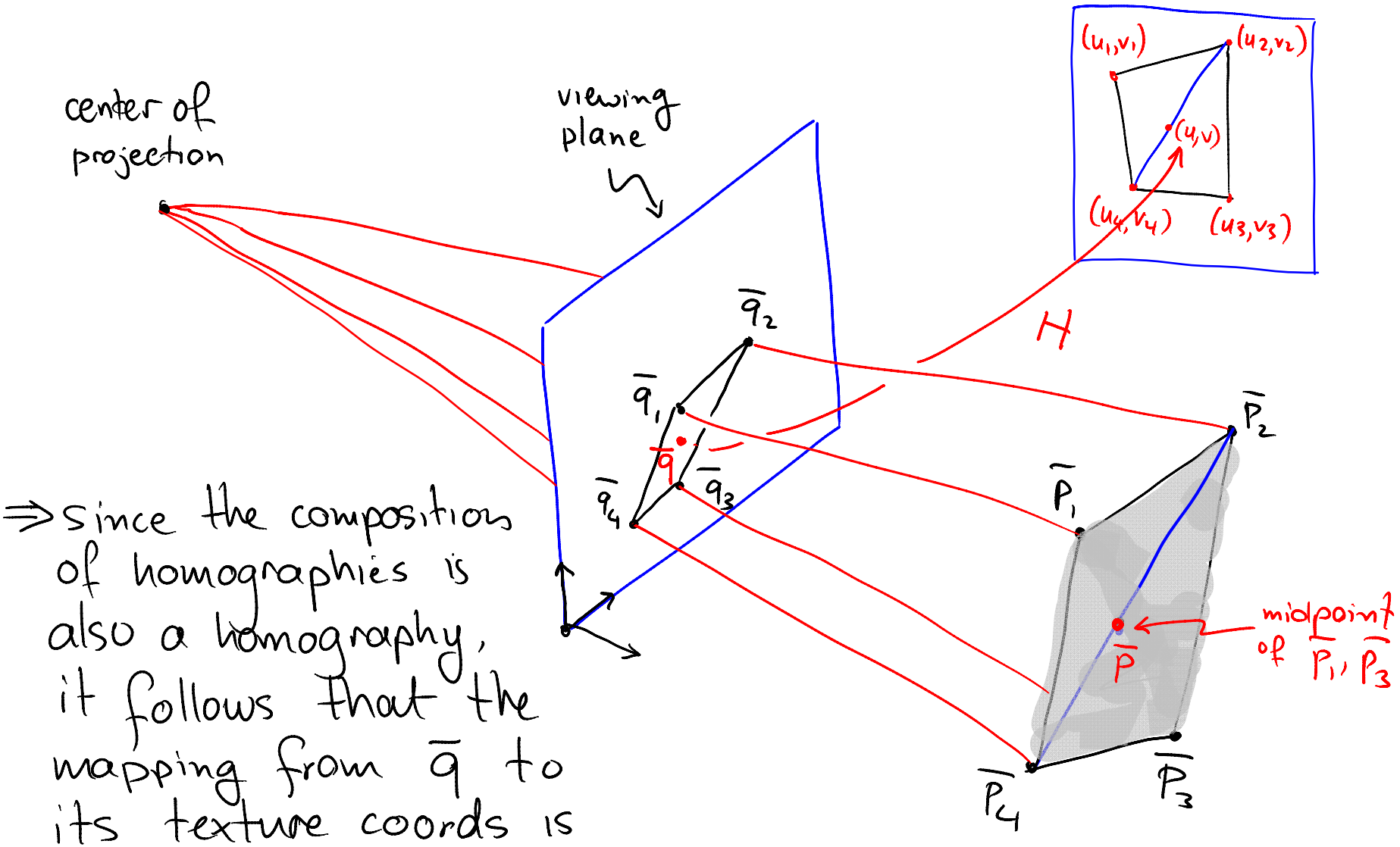
Perspectively-Correct Texture Mapping



Main idea #2

the mapping from polygon points to their texture coordinates is a homography H_2 ↖ most general transform that maps 2D lines to 2D lines

Perspectively-Correct Texture Mapping



$\Rightarrow$ since the composition of homographies is also a homography, it follows that the mapping from $\bar{q}$ to its texture coords is also a homography H

Perspectively-Correct Texture Mapping

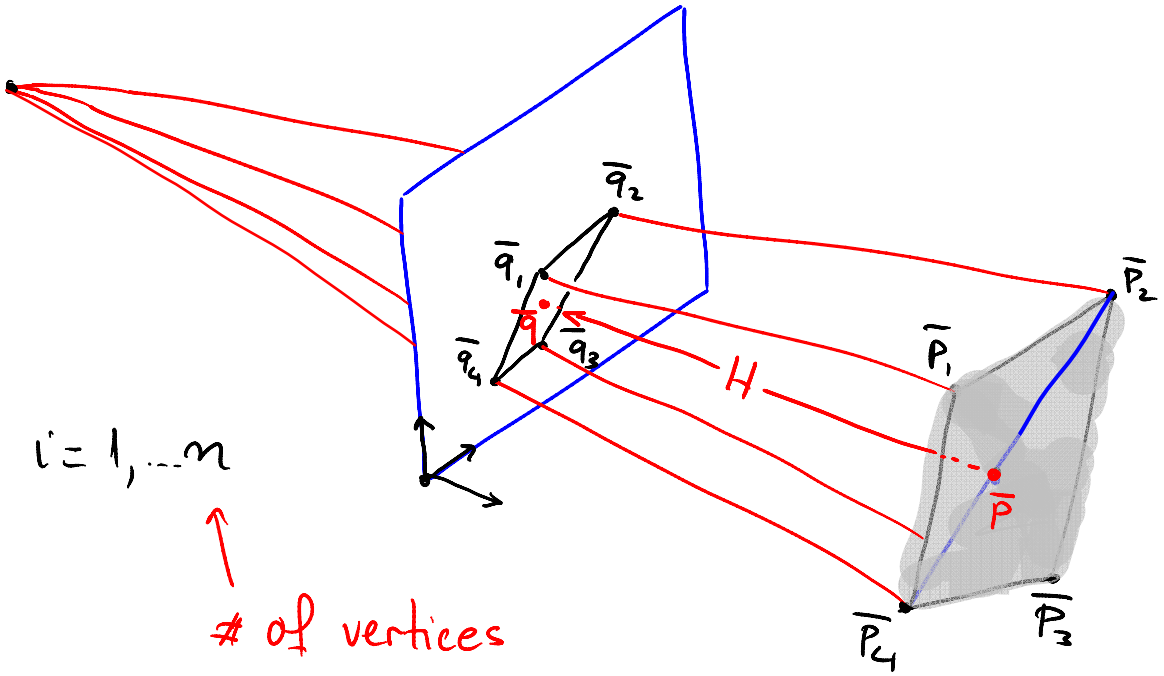
① Compute the homography H such that

$$H\bar{q}_i \cong \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \quad i = 1, \ldots n$$

projection of ith vertex in homogeneous 2D coords

texture coords of ith vertex express as homogeneous vectors

# of vertices



Or look up "rational linear interpolation.

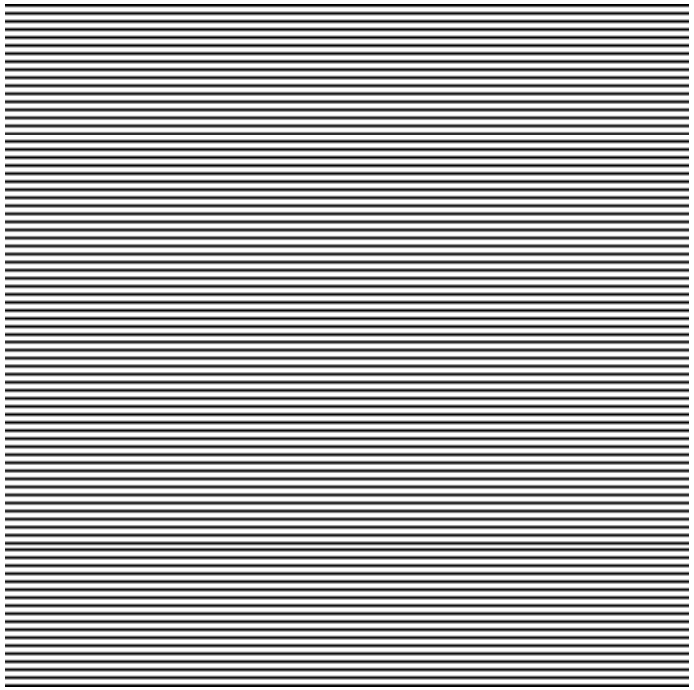
② Use H to compute the texture coords of a pixel $\bar{q} = (x_i, y_i, 1)$ in homogeneous coords

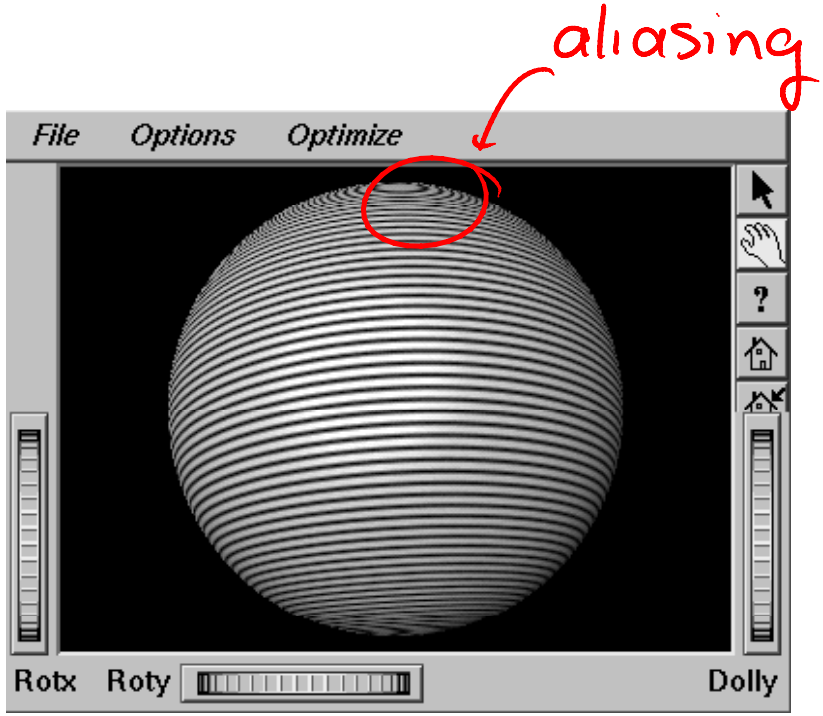
Topic 11:

Texture Mapping

- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

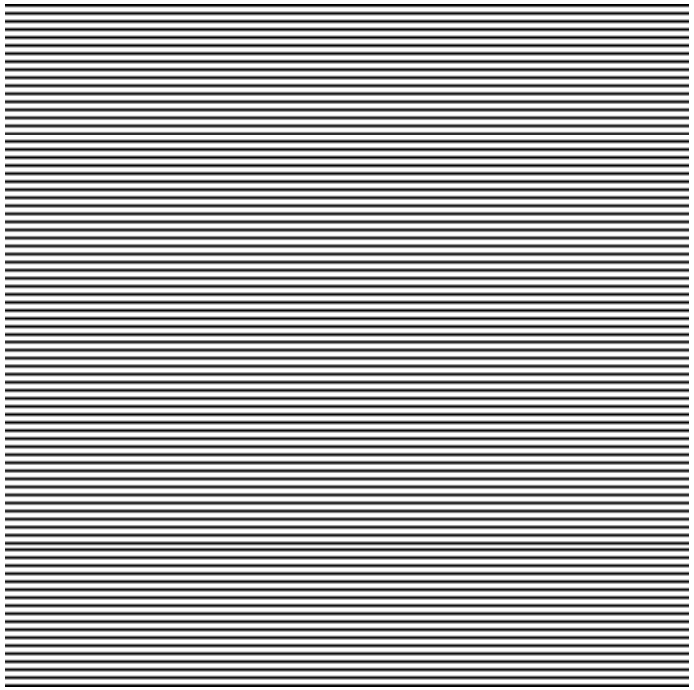
Aliasing During Texture Mapping

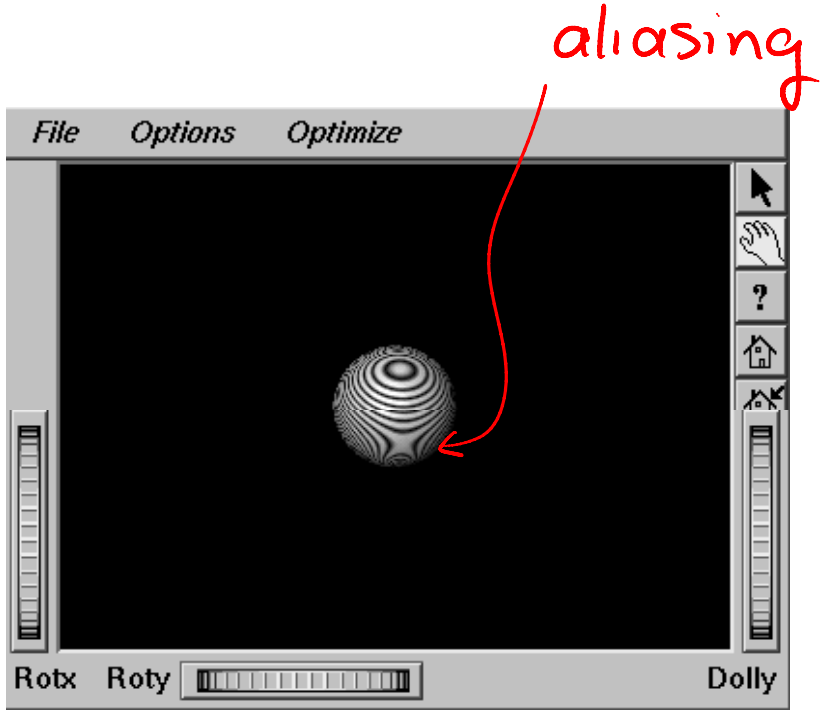




Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies")

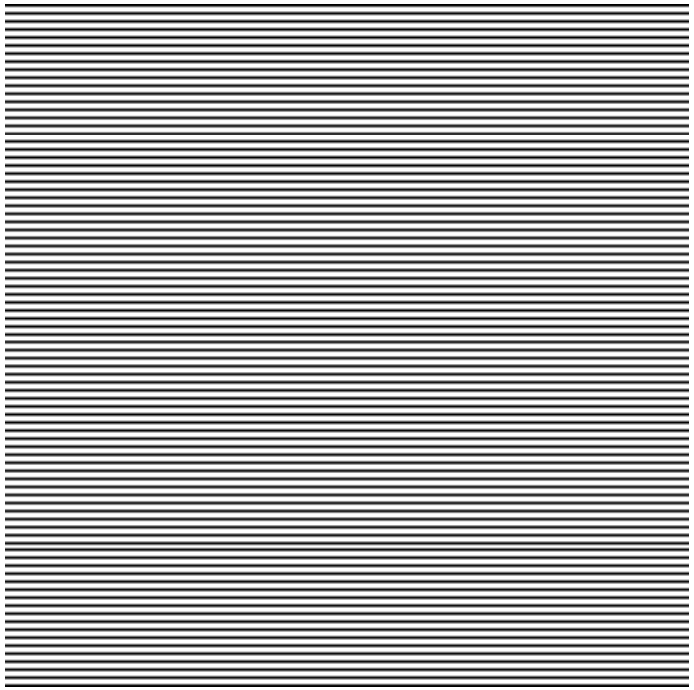
Aliasing During Texture Mapping

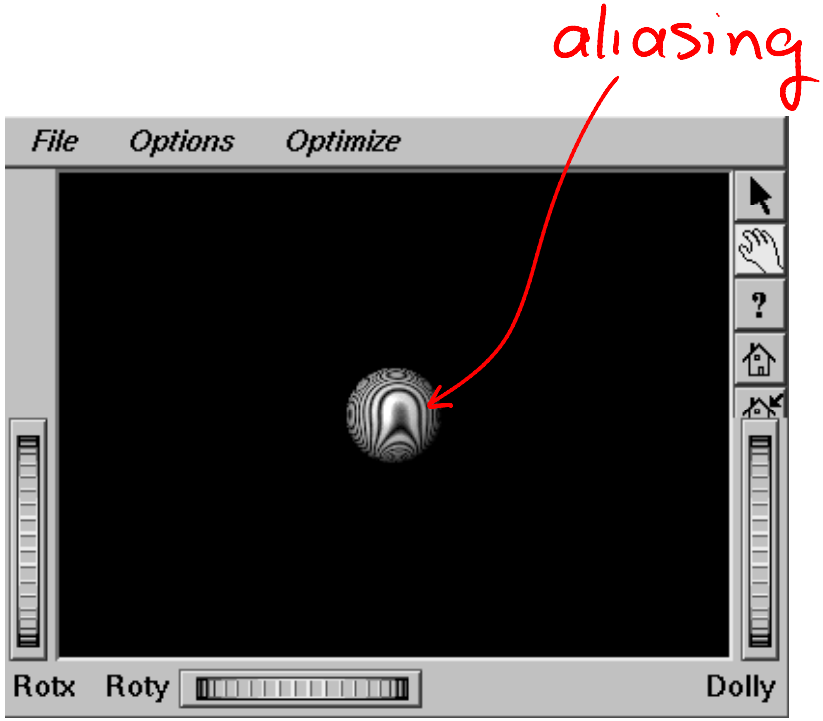




Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies")

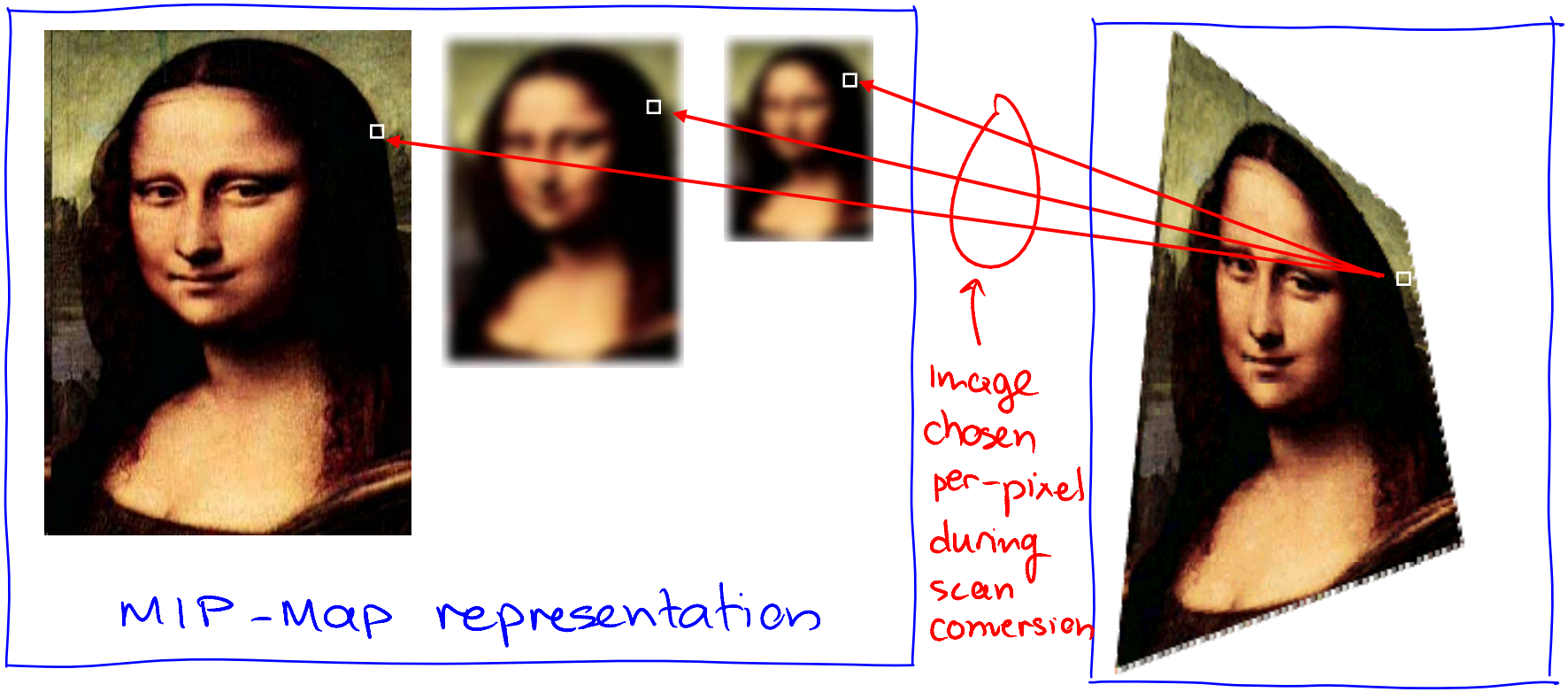
Aliasing During Texture Mapping





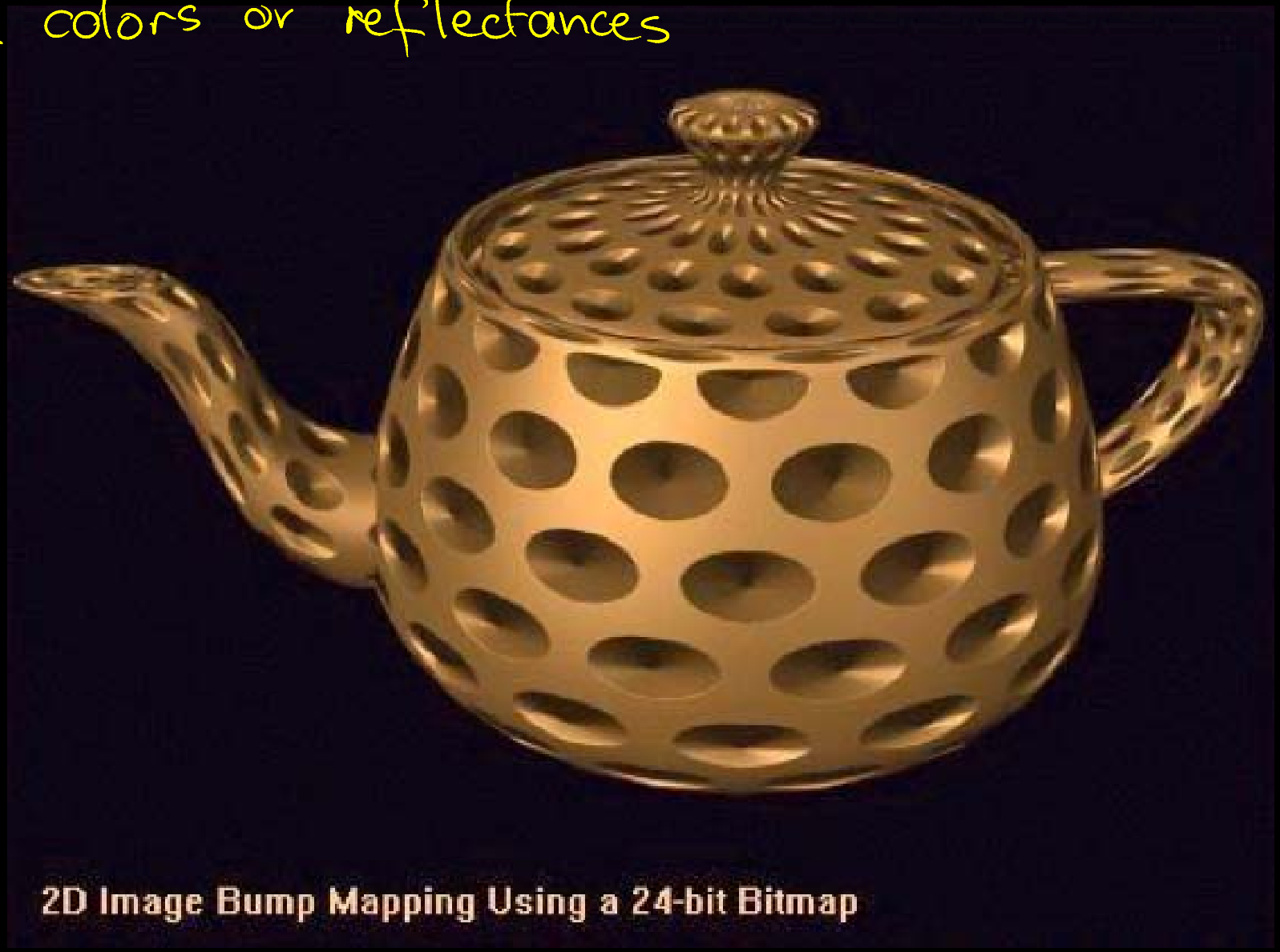
Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies")

MIP-Mapping: Basic Idea

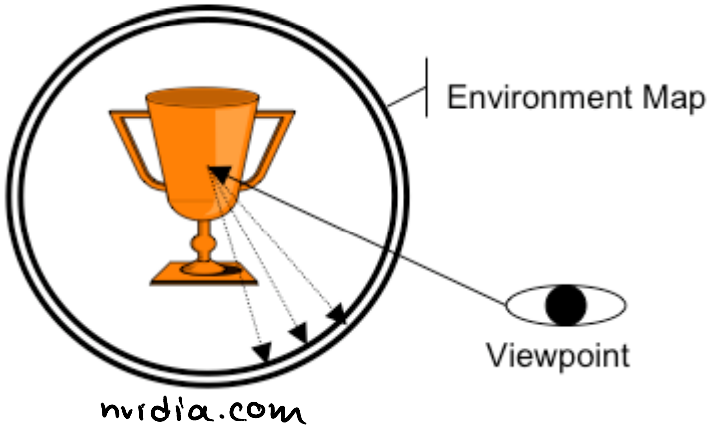


Solution: Use high-resolution texture for rendering objects that are close & low-res (i.e. blurred) texture when they are far away

We can use textures that perturb normals instead of colors or reflectances

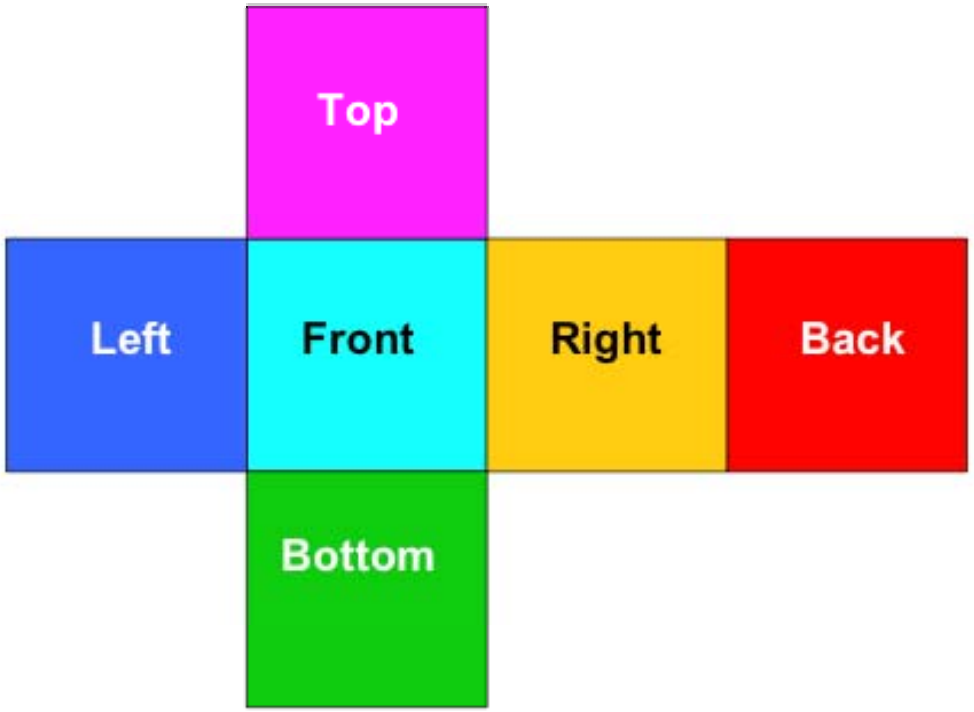


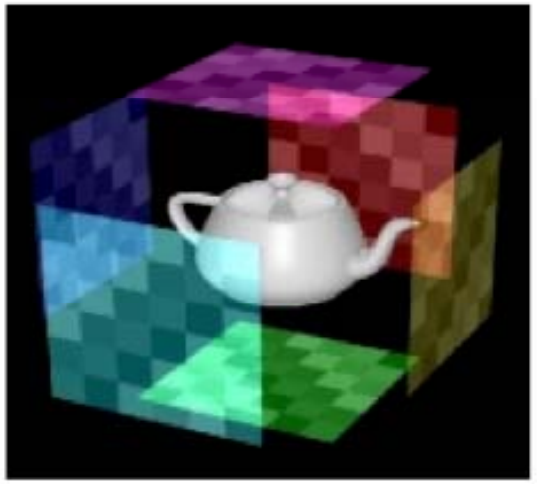
Environment Mapping



- Place the center of projection inside a sphere or cube
- Texture-map the sphere/cube INTERIOR with a photo
- Rendered images now correspond to views of the environment where photo was taken
- Particularly effective technique for rendering reflective objects

Environment Mapping

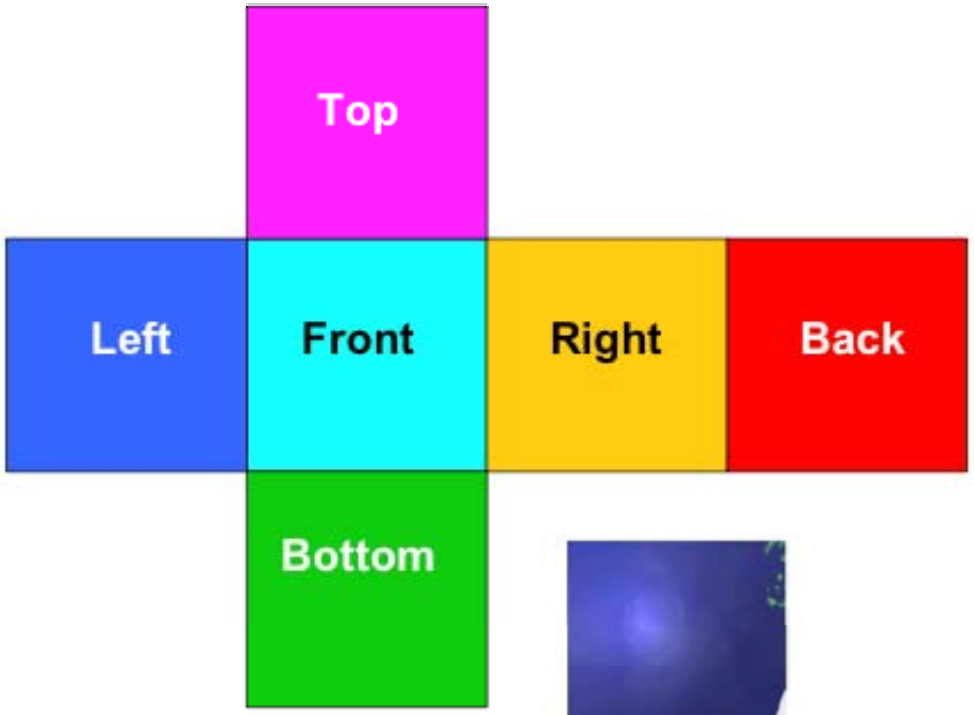


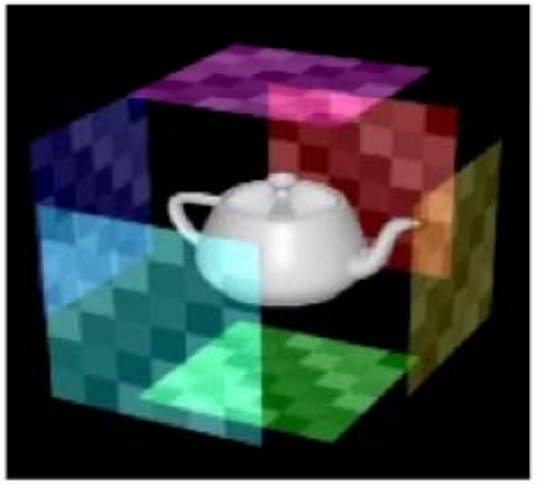


nvidia.com

To create a full 360-degree environment map, we must texture all 6 interior faces of the cube

Environment Mapping





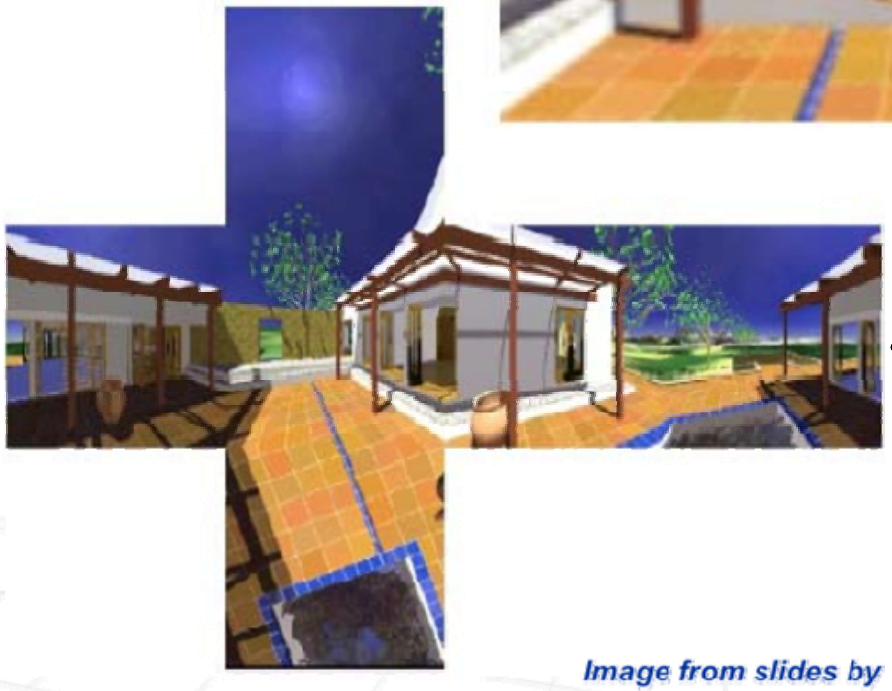
nvidia.com

← texture map

nvidia.com

Environment Mapping

environment mapping result



← texture map

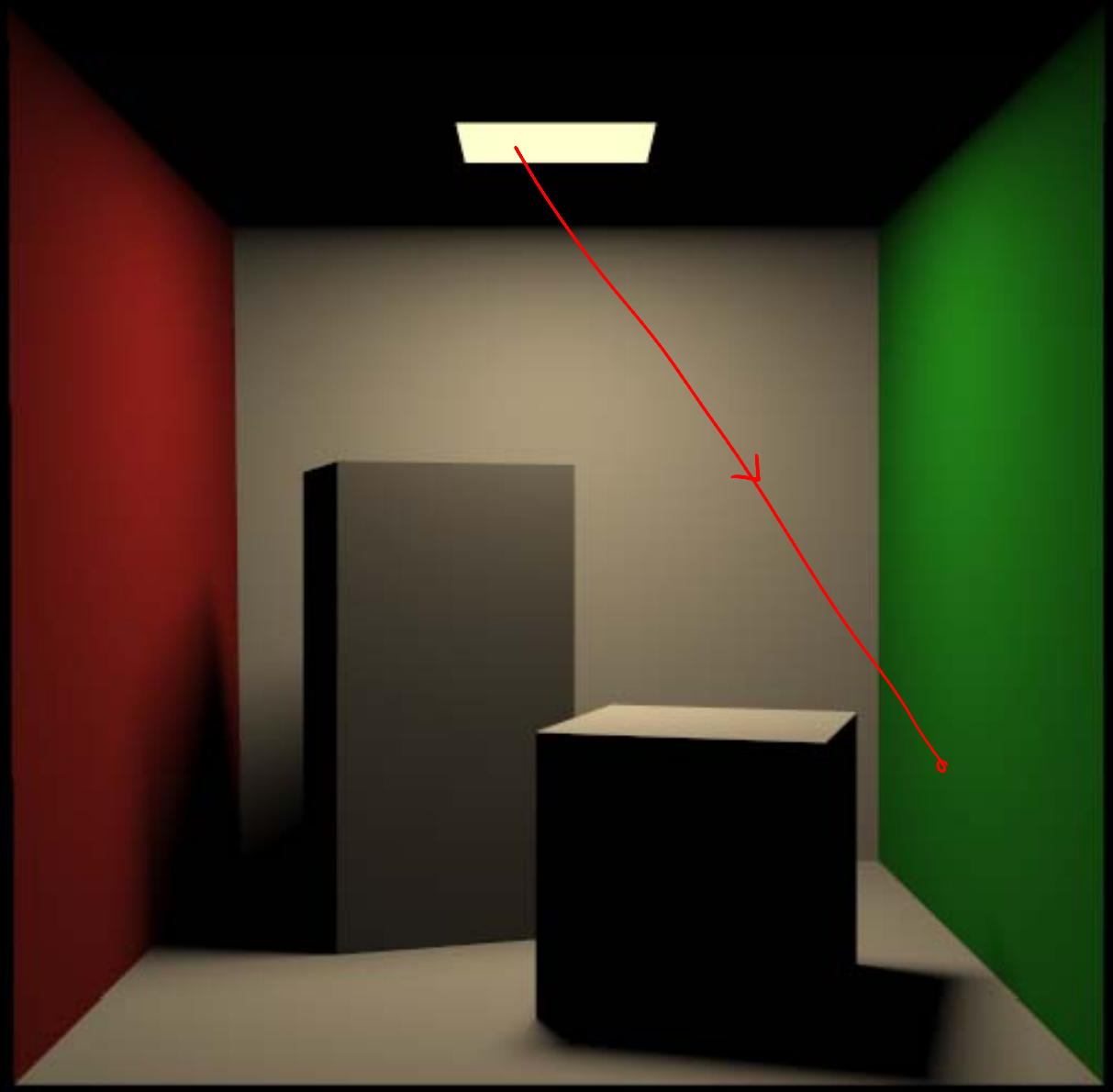
Image from slides by

Topic 12:

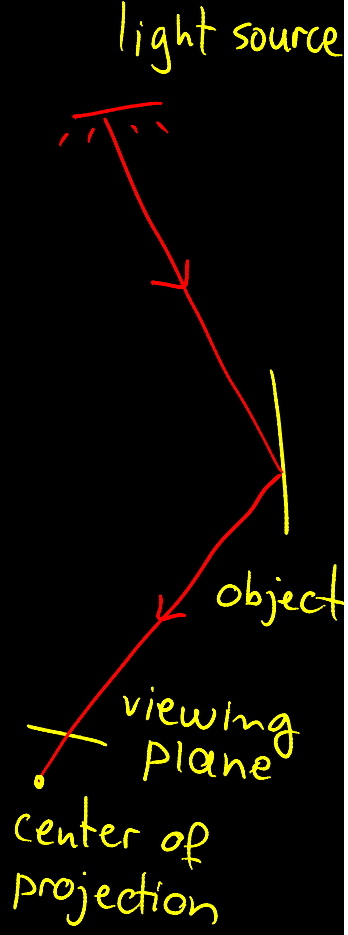
Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

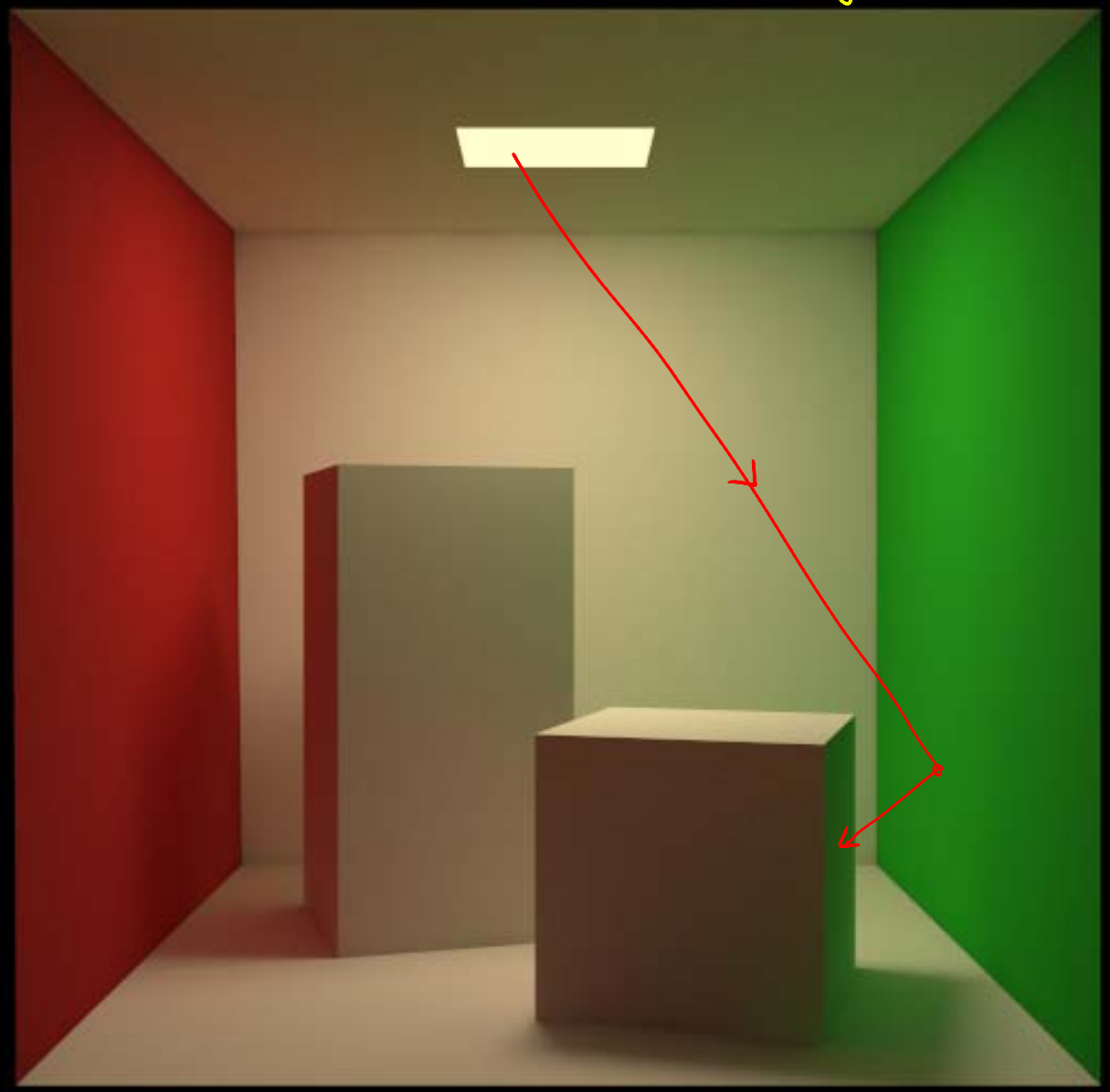
Area Light Source, Direct Lighting



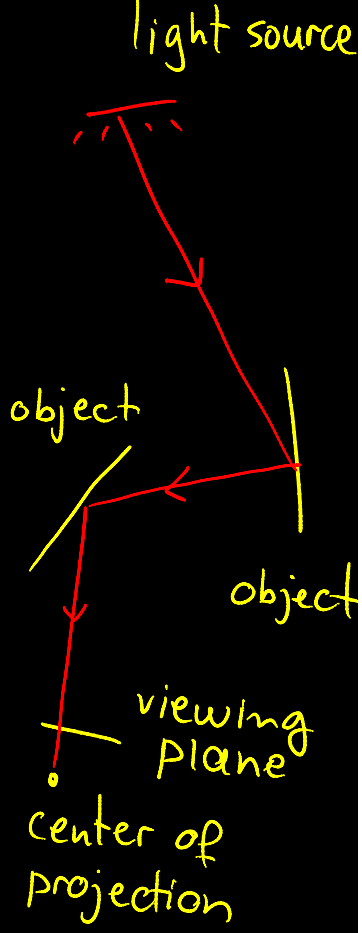
Can be computed using local shading models (eg. Phong)



Area Light Source, Indirect Lighting



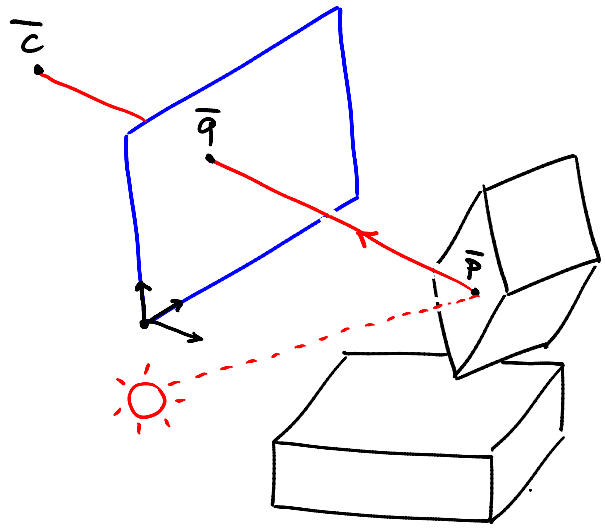
Not possible to produce using local shading models (eg. Phong)



Rasterization vs. Ray Tracing

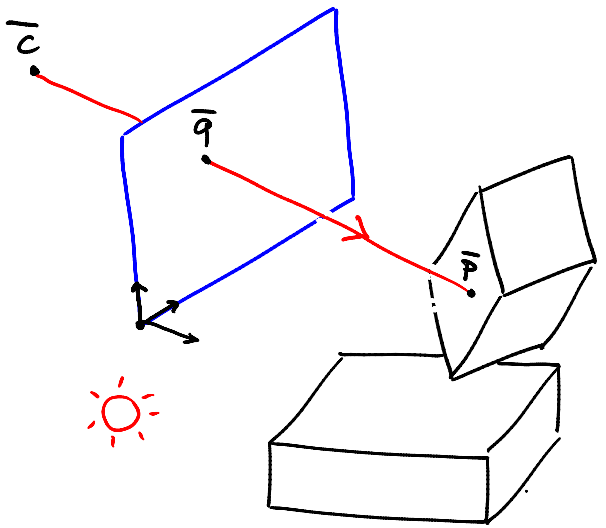
Rasterization:

- Project geometry onto image
- Compute pixel color using local shading model



Ray tracing:

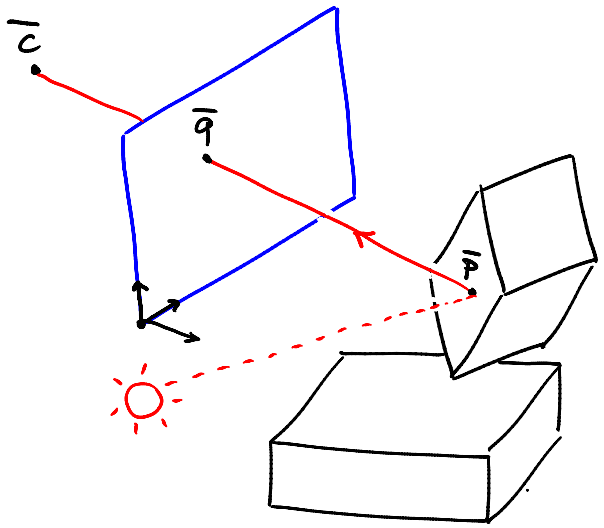
- Project pixels (aka "image samples") backwards onto scene
- Compute pixel color at $\bar{q}$ by estimating light reaching $\bar{p}$ directly or indirectly



Ray Tracing: Basic Idea

Rasterization:

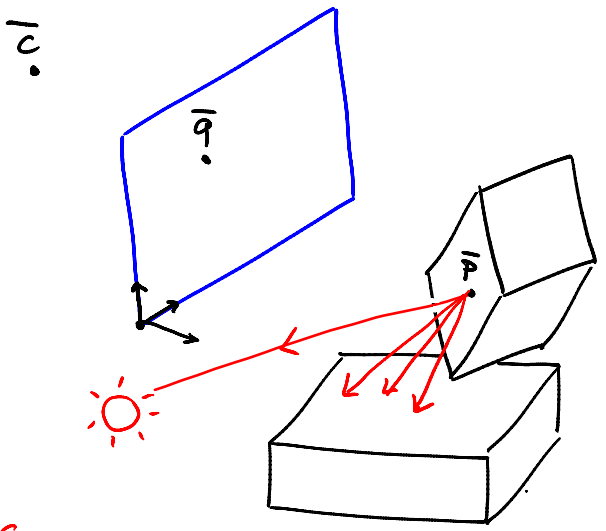
- Project geometry onto image
- Compute pixel color using local shading model



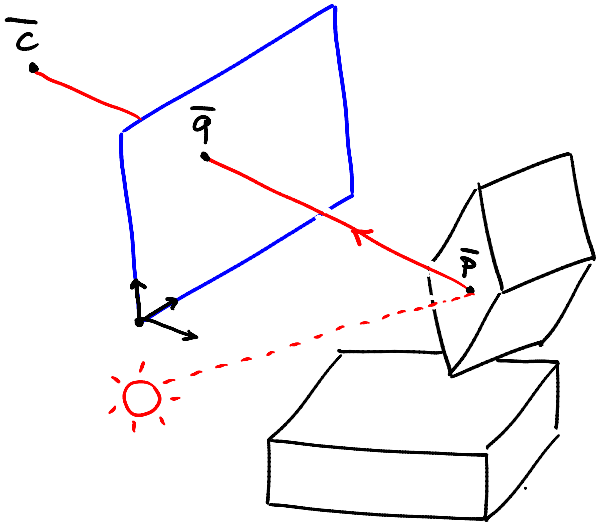
Ray tracing:

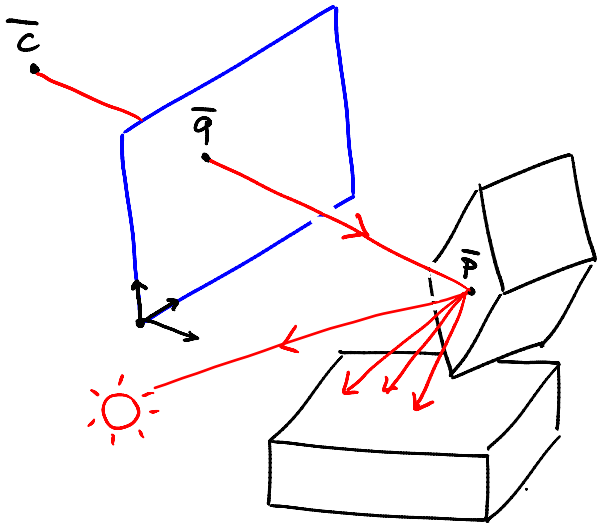
- Project pixels (aka "image samples") backwards onto scene
- Compute pixel color at $\bar{q}$ by estimating light reaching $\bar{p}$ directly or indirectly

  done by recursively casting rays from $\bar{p}$ to possible incident directions



Ray Tracing: Basic Idea

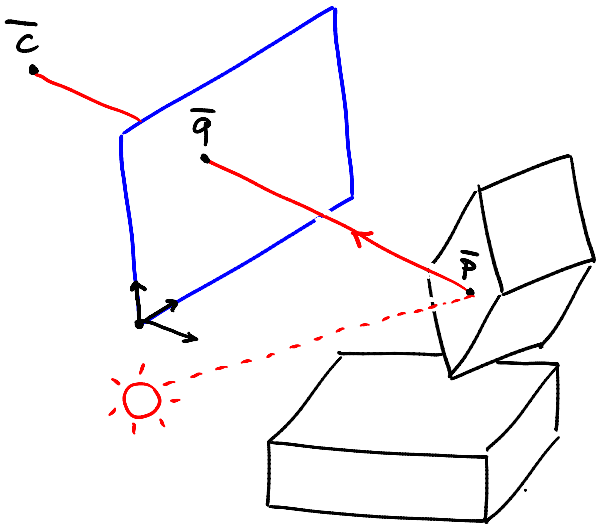


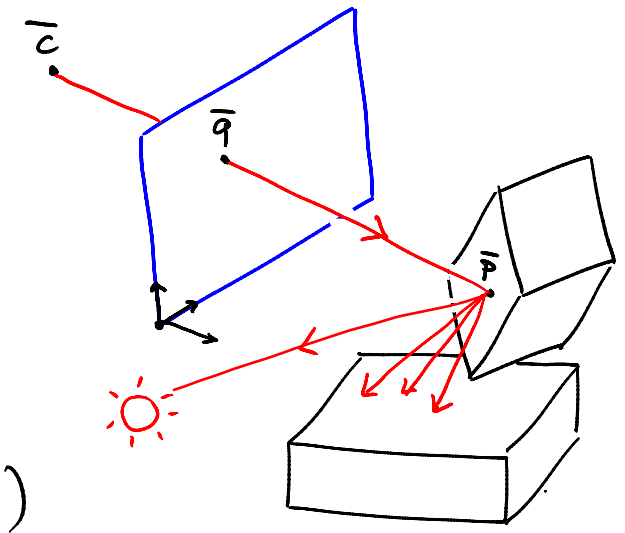


Ray Tracing: Advantages

- highly customizable ("plug-ins" for reflectance models, ray sampling functions)
- can model shadows, arbitrary reflections (eg. mirrors), refractions, indirect illumination, sub-surface scattering, ...
- parallelizable
- allows trading off speed for accuracy (through #cast rays)

...





Ray Tracing: Basic Algorithm

Basic loop:

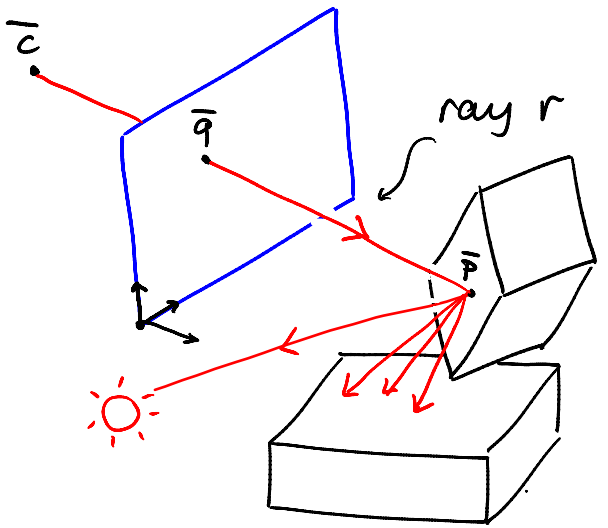
for each pixel $\bar{q}$

① cast ray r through $\bar{q}$

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

③ estimate amount of light reaching $\bar{p}$

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r



Ray Tracing: Basic Algorithm

Basic loop:

for each pixel $\bar{q}$

① cast ray r through $\bar{q}$

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

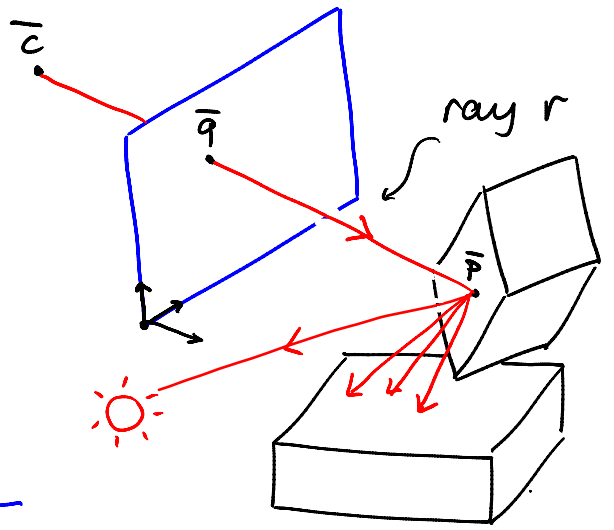
③ estimate amount of light reaching $\bar{p}$

a. "spawn" rays $r_1, r_2, \ldots, r_k$ from $\bar{p}$ in various directions

b. if ray r_i hits a light source, estimate light travelling along r_i and <u>stop</u>

c. else apply loop recursively to ray r_i

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r

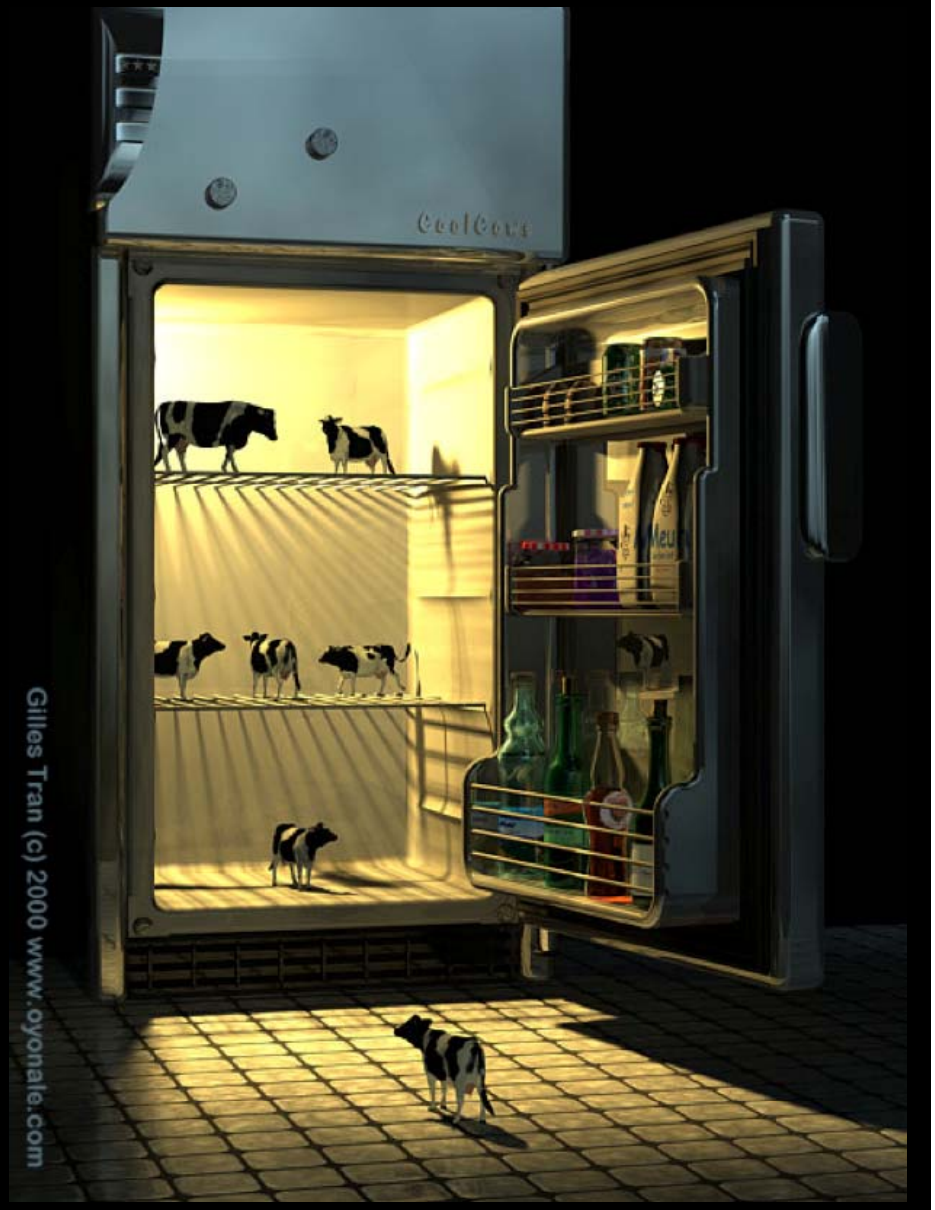


Ray tracing in the movies

Christensen et al, 2006

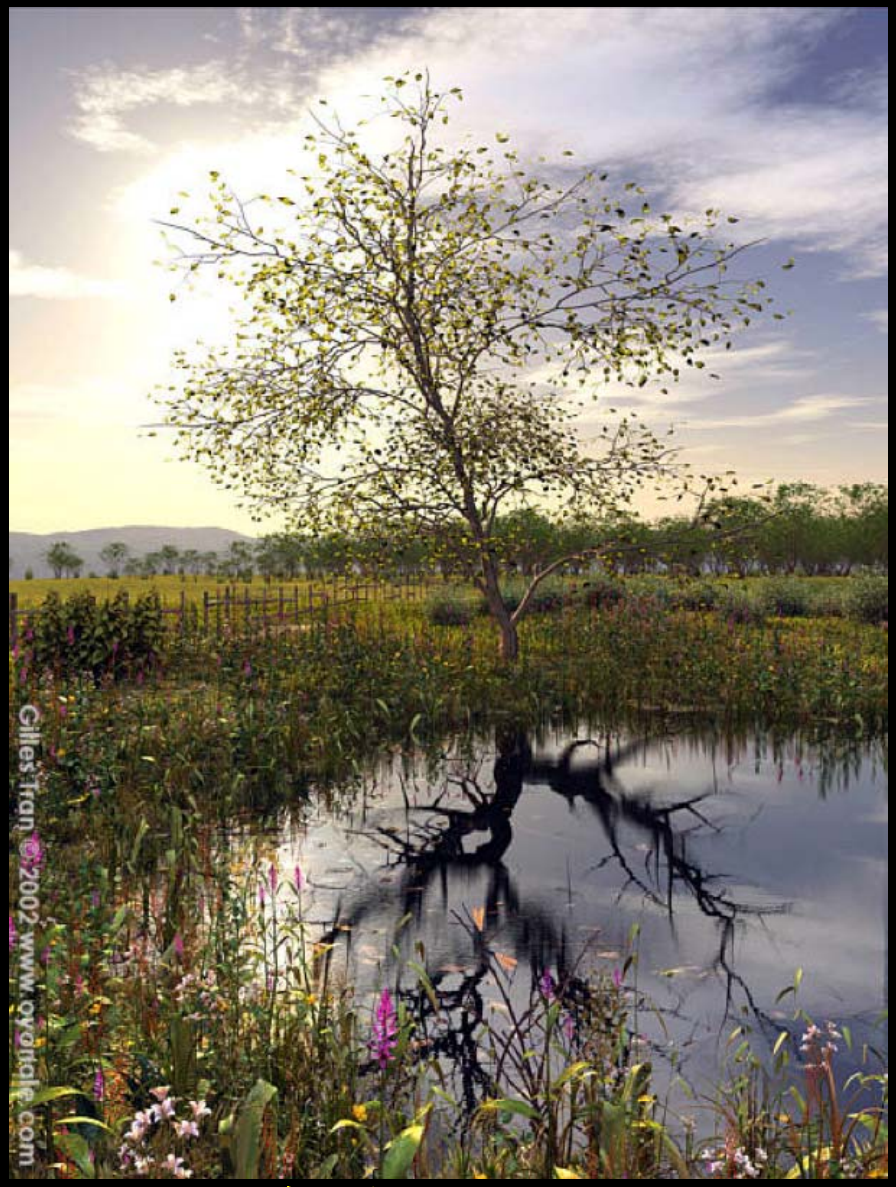
www.povray.org/community/hof

"Main steet (blue)"



"The Cool Cows"

www.povray.org/community/hof



"The Dark Side of Trees"

"Capriccio" G. Obukhov et al, 2003

Online Ray Tracing Competitions

www.irtc.org/stills

380K triangles, 104 lights, full global illumination in real time

SIGGRAPH'05 Course by Slusallek et al

15 cars, 240 Mquads, 80M rays

Render time : without optimizations > 4 days
with optimizations ~ 6 hrs

Computational Issues in Basic Ray Tracing

Basic loop:

for each pixel $\bar{q}$

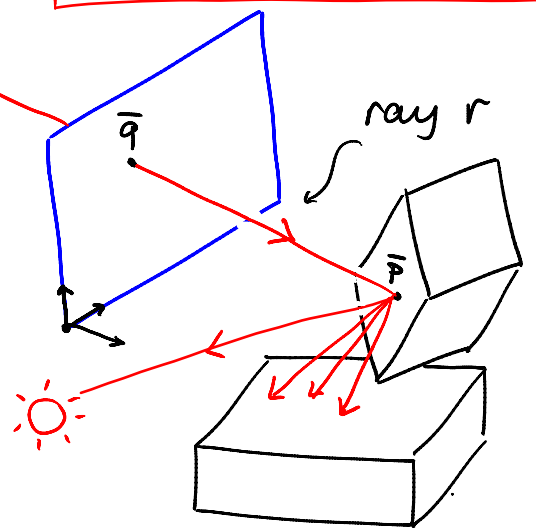
1. cast ray r through $\bar{q}$
2. find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)
3. estimate amount of light reaching $\bar{p}$
   a. "spawn" rays $r_1, r_2, \ldots, r_k$ from $\bar{p}$ in various directions
   b. if ray r_i hits a light source, estimate light travelling along r_i and stop
   c. else apply loop recursively to ray r_i
4. estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r

defining the ray r

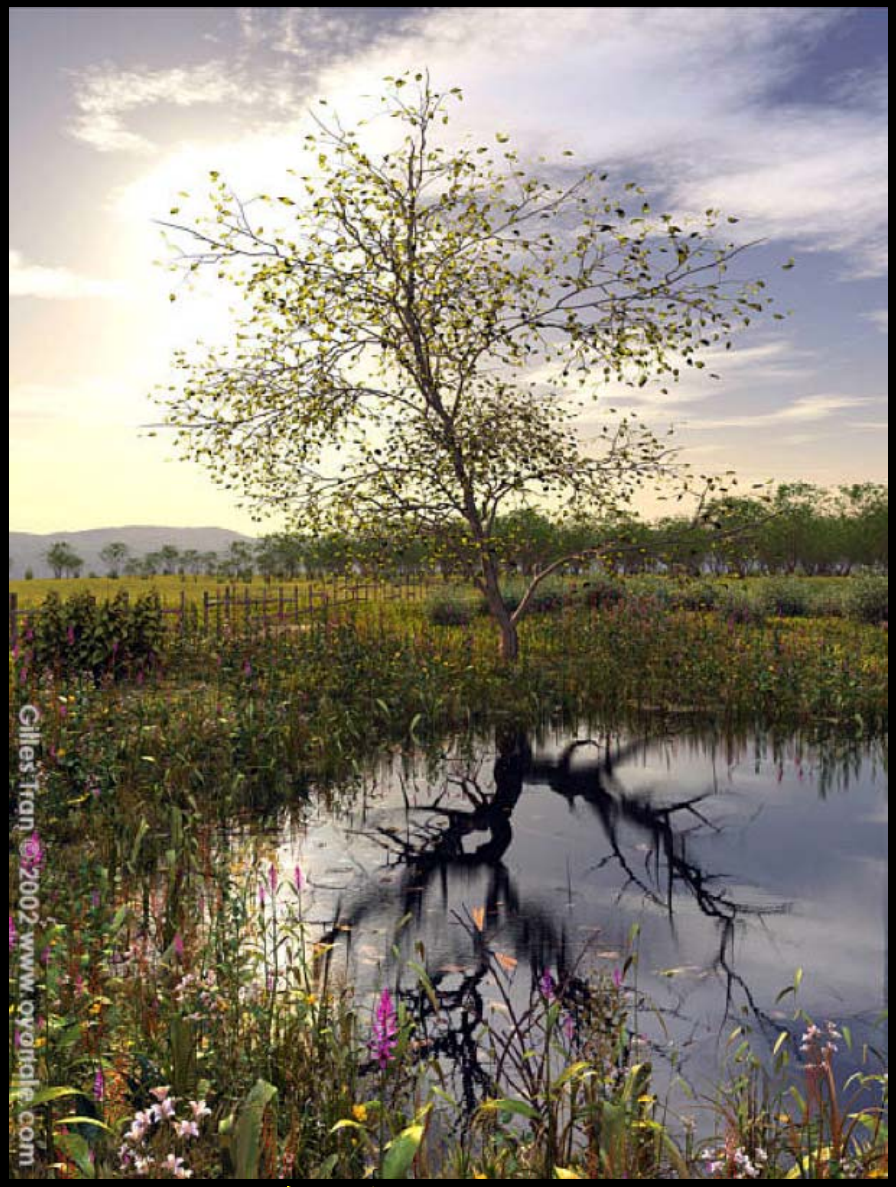
computing ray-scene intersections

evaluating reflectance model at $\bar{p}$

spawning rays



www.povray.org/community/hof



"The Dark Side of Trees"

"Capriccio" G. Obukhov et al, 2003

Ray Tracing: Basic Algorithm

Basic loop:

for each pixel $\bar{q}$

① cast ray r through $\bar{q}$

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

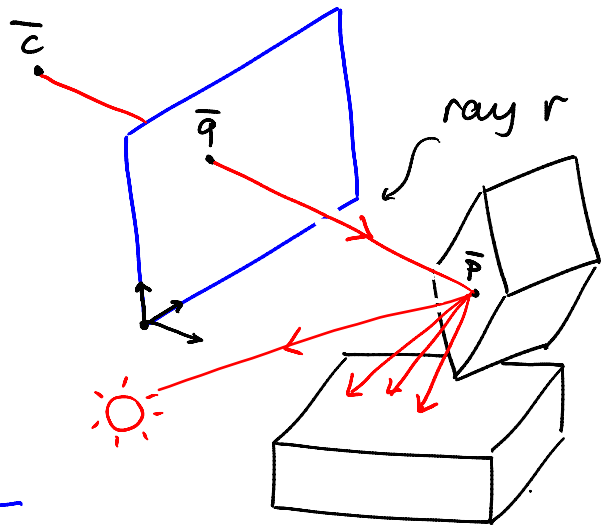
③ estimate amount of light reaching $\bar{p}$

a. "spawn" rays $r_1, r_2, \ldots, r_k$ from $\bar{p}$ in various directions

b. if ray r_i hits a light source, estimate light travelling along r_i and <u>stop</u>

c. else apply loop recursively to ray r_i

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r



Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Computing the Ray Through a Pixel

Basic loop:

for each pixel $\bar{q}$

① cast ray r through $\bar{q}$ (defining the ray r)

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

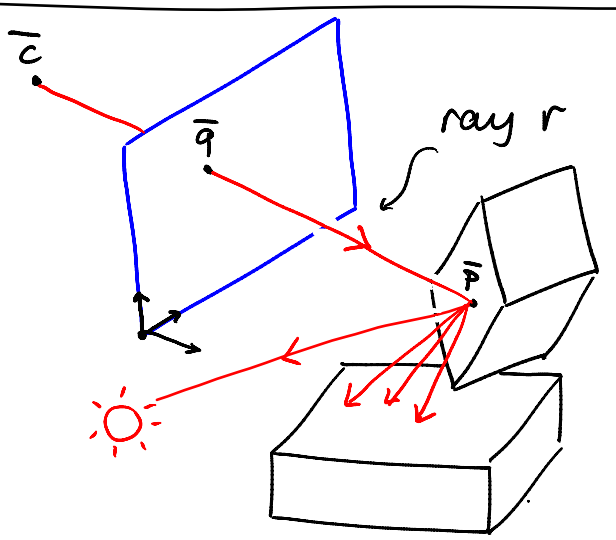
③ estimate amount of light reaching $\bar{p}$

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r

---

Compute ray r given

- discrete pixel position
- world-to-camera matrix M_{wc}
- camera-to-canonical view matrix M_{cv}



Computing the Ray Through a Pixel: Main Idea

Idea: ray through $\bar{q}$ contains the points

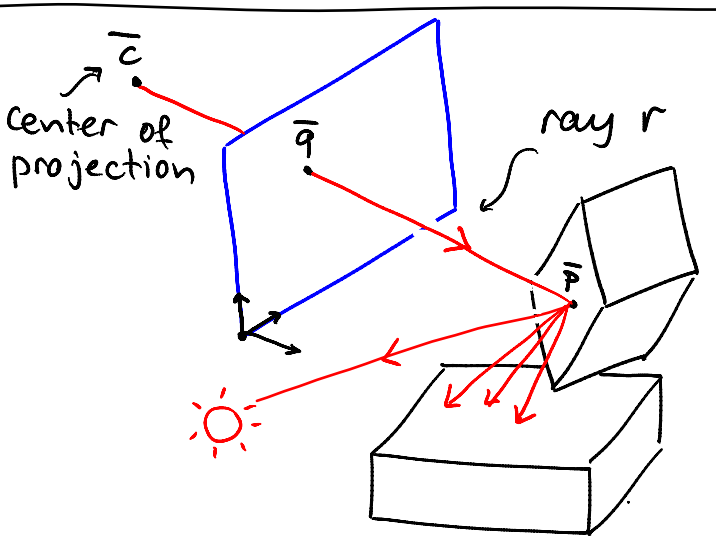
$$\bar{c} + \lambda(\bar{q}_w - \bar{c}) \quad \lambda \in \mathbb{R}$$

where $\bar{q}_w$ are the world coordinates of pixel $\bar{q}$

↖ how do we compute $\bar{q}_w$?

Compute ray r given

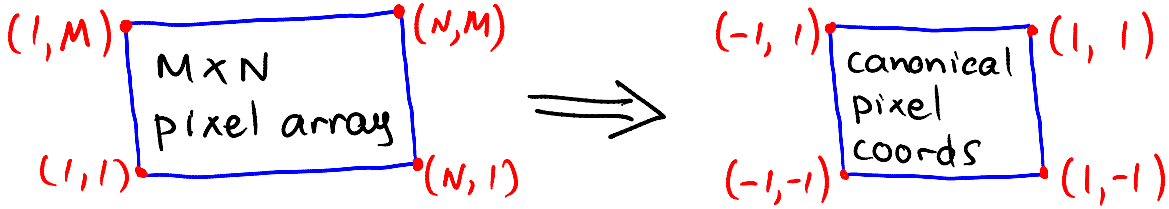
- discrete pixel position
- world-to-camera matrix M_{wc}
- camera-to-canonical view matrix M_{cv}



Computing the Ray Through a Pixel: Steps

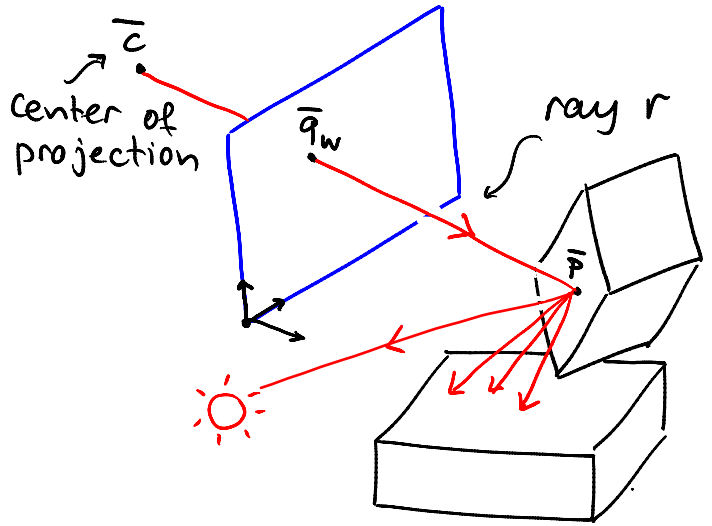
Computing homogeneous coords of $\bar{q}_w$:

① Convert discrete pixel coordinates (row & column number) to canonical coordinates (x_i, y_i) that lie in the range $[-1, 1]$



Compute ray r given

- discrete pixel position
- world-to-camera matrix M_{wc}
- camera-to-canonical view matrix M_{cv}



Computing the Ray Through a Pixel: Steps

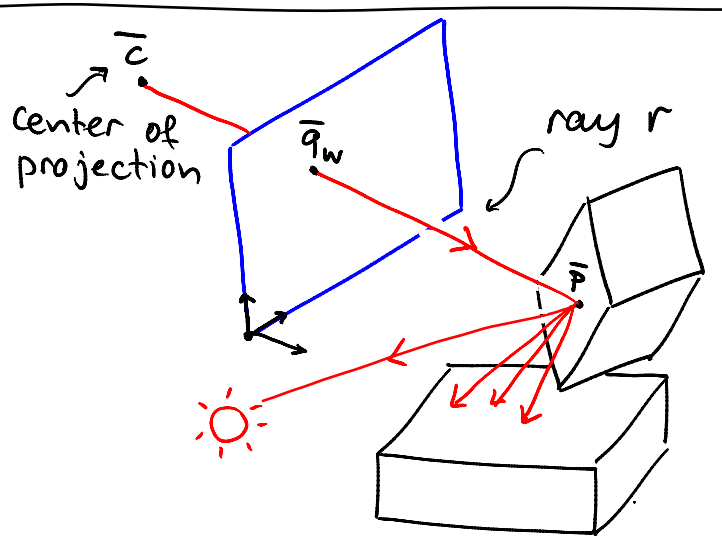
Computing homogeneous coords of $\bar{q}_w$:

1. Compute canonical view coordinates (x_i, y_i)
2. Compute homogeneous 3D canonical view coordinates of $\bar{q}$:

$$\bar{q}_v = \begin{bmatrix} x_i \\ y_i \\ 0 \\ 1 \end{bmatrix}$$

Compute ray r given

- discrete pixel position
- world-to-camera matrix M_{wc}
- camera-to-canonical view matrix M_{cv}



Computing the Ray Through a Pixel: Steps

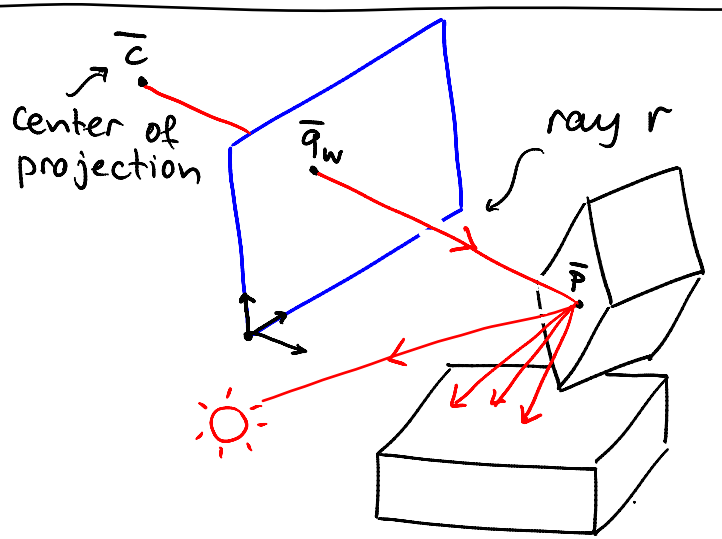
Computing homogeneous coords of $\bar{q}_w$:

1. Compute canonical view coordinates (x_i, y_i)
2. Compute homogeneous 3D canonical view coordinates, $\bar{q}_v$
3. Convert to world coordinates:

$$\bar{q}_w = M_{wc}^{-1} \cdot M_{cv}^{-1} \bar{q}_v$$

Compute ray r given

- discrete pixel position
- world-to-camera matrix M_{wc}
- camera-to-canonical view matrix M_{cv}



Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Computing Ray-Object Intersections

Basic loop:

for each pixel $\bar{q}$

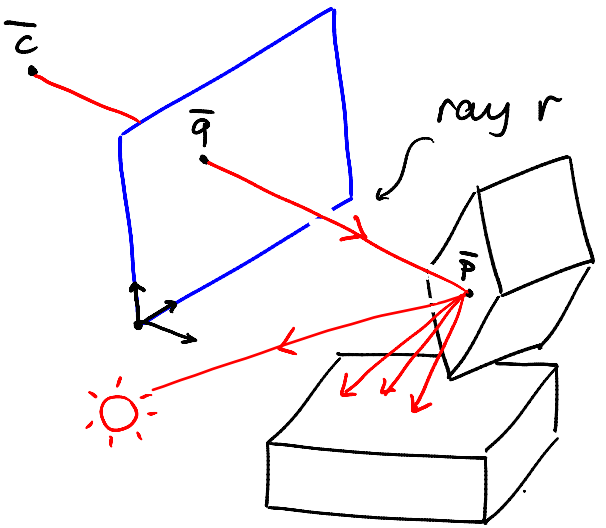
① cast ray r through $\bar{q}$

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

③ estimate amount of light reaching $\bar{p}$

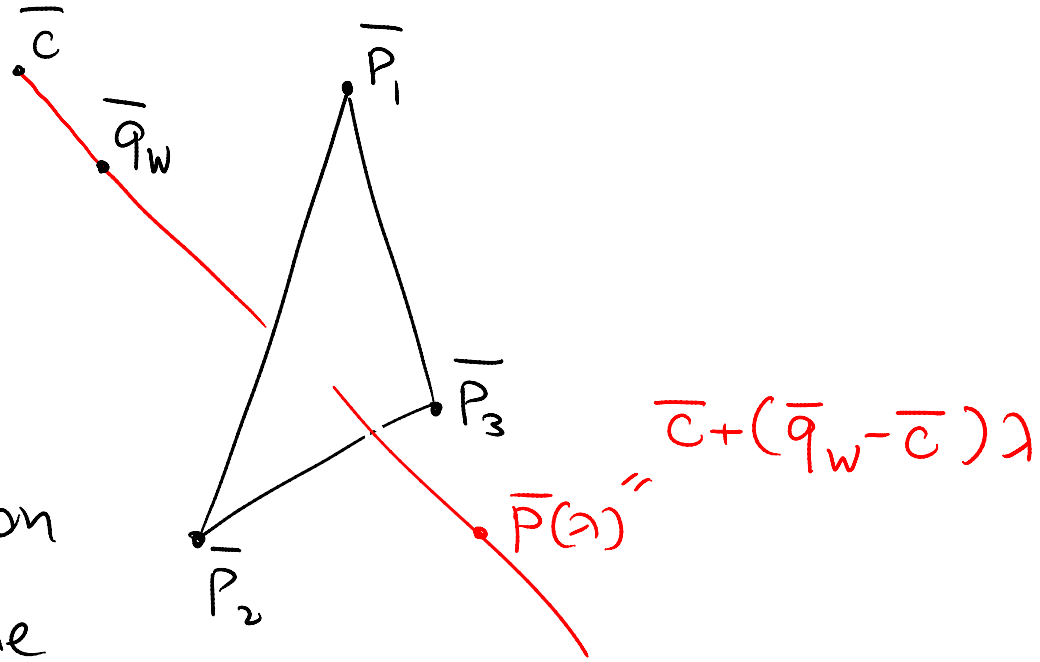
computing ray-scene intersections

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r



Computing Ray-Triangle Intersections

Algorithm #1



(a.) compute normal

$$\vec{n} = (\bar{P}_2-\bar{P}_1)\times(\bar{P}_3-\bar{P}_1)$$

(b.) compute intersection of ray and plane of triangle

i.e. find λ^* that satisfies

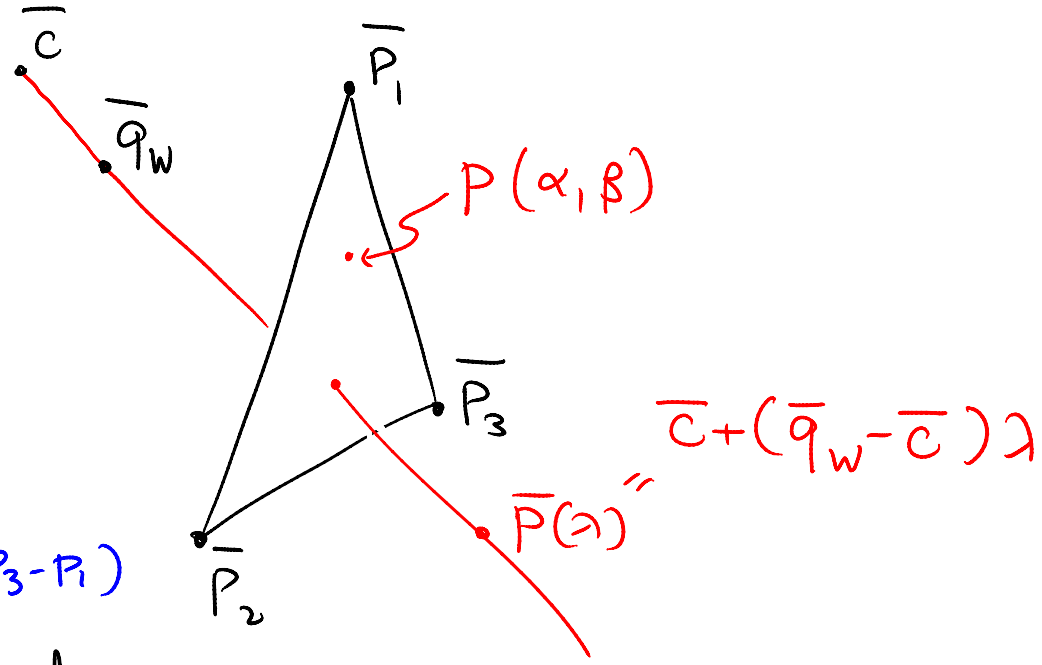
$$\left[\bar{P}_1 - \bar{p}(\lambda^*)\right]\cdot\vec{n} = 0$$

(c.) verify that $p(\lambda^*)$ falls within triangle

e.g. using half-space constraints (assignment 1)

Computing Ray-Triangle Intersections

Algorithm #2



(a.) Parameterize the triangle plane

$p(\alpha,\beta) =$

$p_1 + \alpha(p_2-p_1) + \beta(p_3-p_1)$

(b.) Find α, β, λ^* that satisfy $p(\alpha,\beta) = p(\lambda^*)$

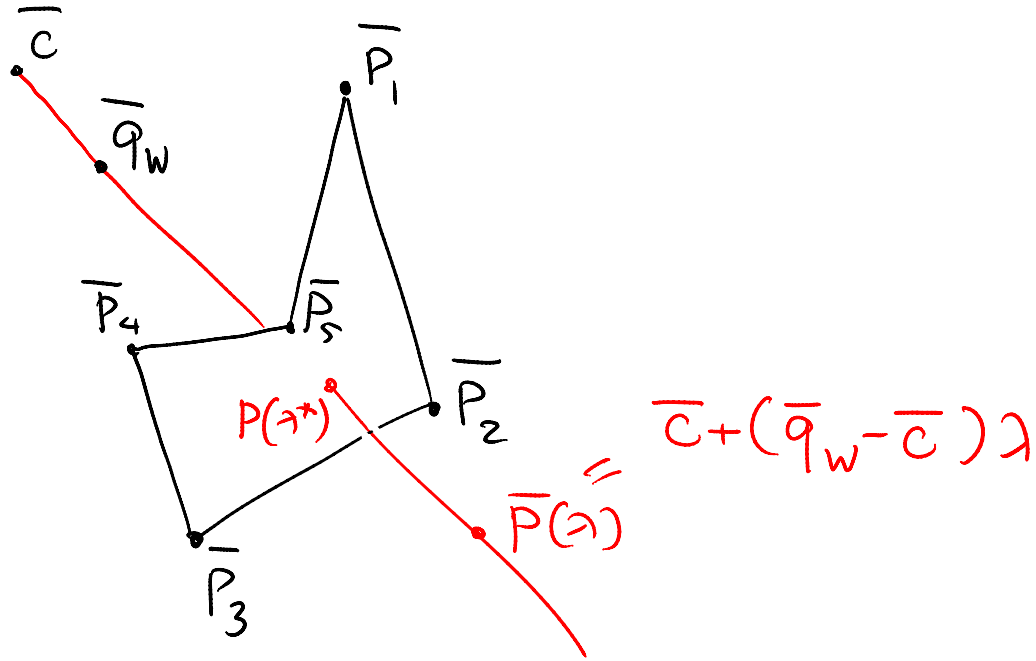
vectors expressed in euclidean 3D coords; unknowns

$$\Rightarrow \text{solve} \quad \underbrace{\left[-(\bar{p}_2-\bar{p}_1) \;\; -(\bar{p}_3-\bar{p}_1) \;\; (\bar{q}_w-\bar{c})\right]}_{3\times 3 \text{ matrix}} \begin{bmatrix}\alpha\\ \beta\\ \lambda^*\end{bmatrix} = \bar{p}_1 - \bar{c}$$

(c.) $p(\lambda^*)$ inside triangle $\Longleftrightarrow$ $\alpha \geq 0,\ \beta \geq 0,\ \alpha+\beta \leq 1$

Computing Ray-Polygon Intersections

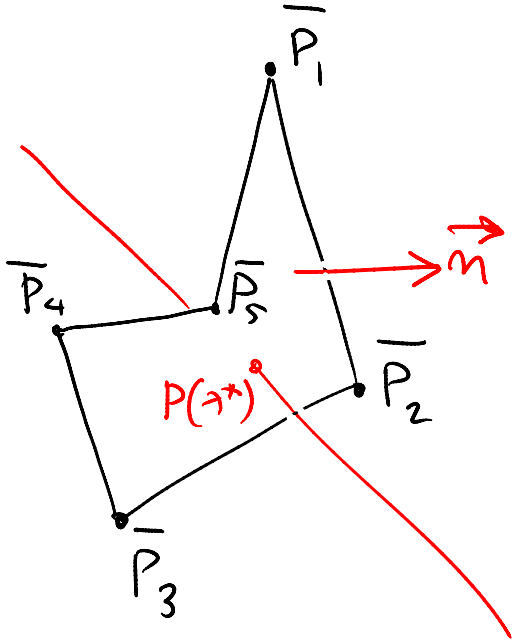
(a.) Compute $\bar{p}(\lambda^*)$ using Algorithm 1 or 2 (by picking 3 adjacent non-collinear vertices)

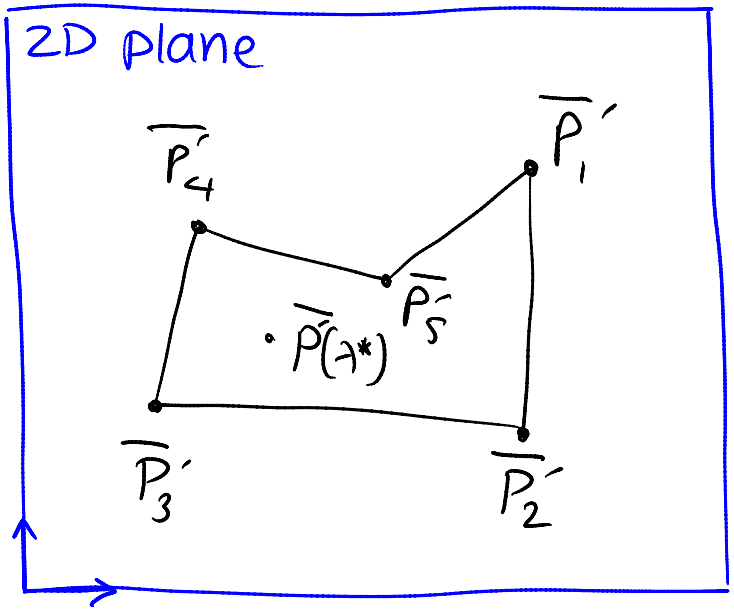


(b.) Verify that $p(\lambda^*)$ lies inside the polygon

- Convert the 3D polygon & $p(\lambda^*)$ to 2D
- Do the verification in 2D (recall assignment #1)

Computing Ray-Poly Intersections: Step a





- Suppose $\vec{n}$ is not along z-axis
- Project all vertices and $p^*(\lambda)$ onto xy-plane:

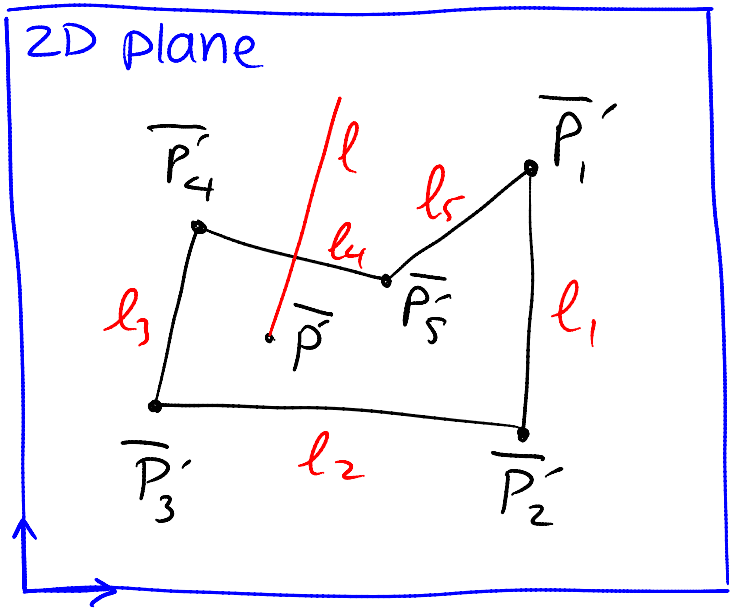
$$\bar{P}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \bar{P}_i'$$

if $\vec{n}$ is along z axis, project onto xz-plane

Computing Ray-Poly Intersections: Step b

Key theorem:

If $\bar{p}'$ inside, every 2D half-line starting at $\bar{p}'$ must intersect the polygon's boundary an <u>odd</u> # of times



Verification algorithm:

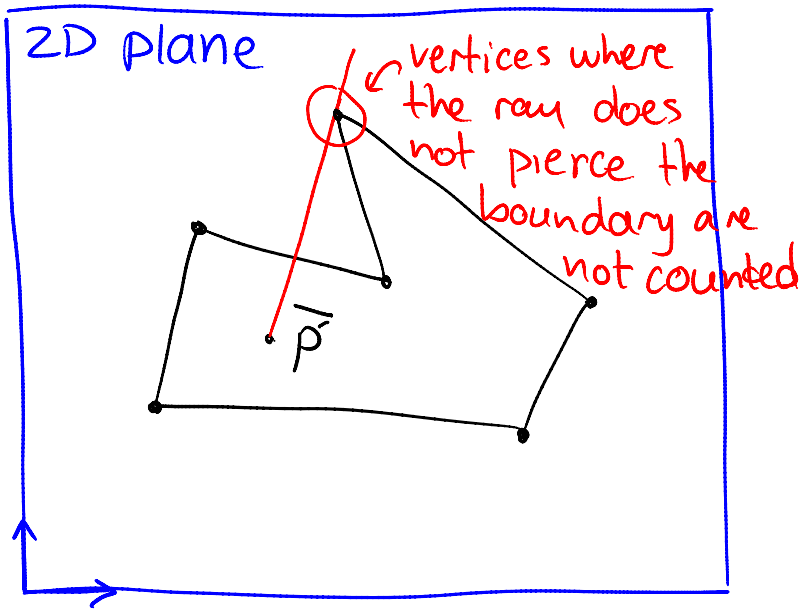
eg. $\bar{q}' = \frac{1}{2}(\bar{P}_1' + \bar{P}_2')$

① pick any non-vertex point on boundary
② define lines ℓ through $\bar{p}', \bar{q}'$ and l_i through $\bar{P}_i', \bar{P}_{i+1}'$
③ intersect ℓ with each l_i
④ count intersections that are
⑤ on same side of $\bar{p}'$, and
⑥ on polygon boundary

Computing Ray-Poly Intersections: Step b

Key theorem:

If $\bar{p}'$ inside, every 2D half-line starting at $\bar{p}'$ must intersect the polygon's boundary an <u>odd</u> # of times

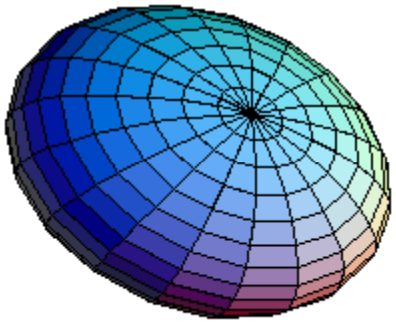


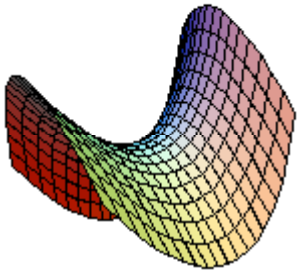
Verification algorithm:

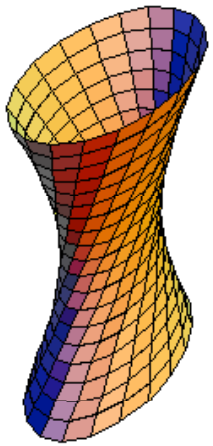
eg. $\bar{q}' = \frac{1}{2}(\bar{p}_1' + \bar{p}_2')$

1. pick any non-vertex point on boundary
2. define lines ℓ through $\bar{p}', \bar{q}'$ and l_i through $\bar{p}_i', \bar{p}_{i+1}'$
3. intersect ℓ with each l_i
4. count intersections ← counting is a bit more involved if line ℓ intersects a polygon vertex

Computing Ray-Quadric Intersections



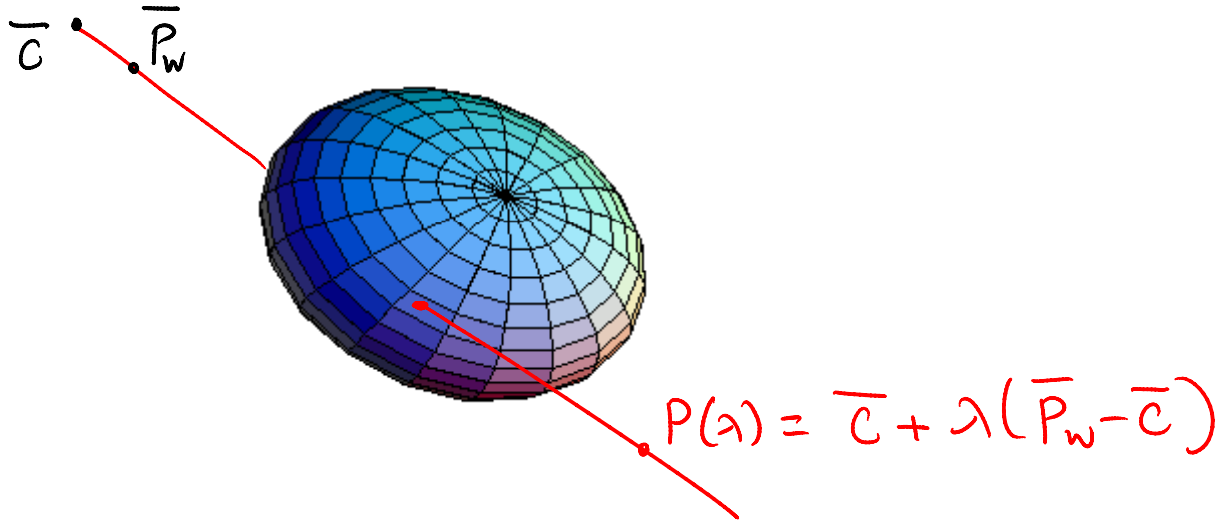




General implicit equation

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \underbrace{\begin{bmatrix} A & D & E & G \\ D & B & F & H \\ E & F & C & I \\ G & H & I & J \end{bmatrix}}_{\text{defined up to a scale factor}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

Computing Ray-Quadric Intersections



Must solve the equation

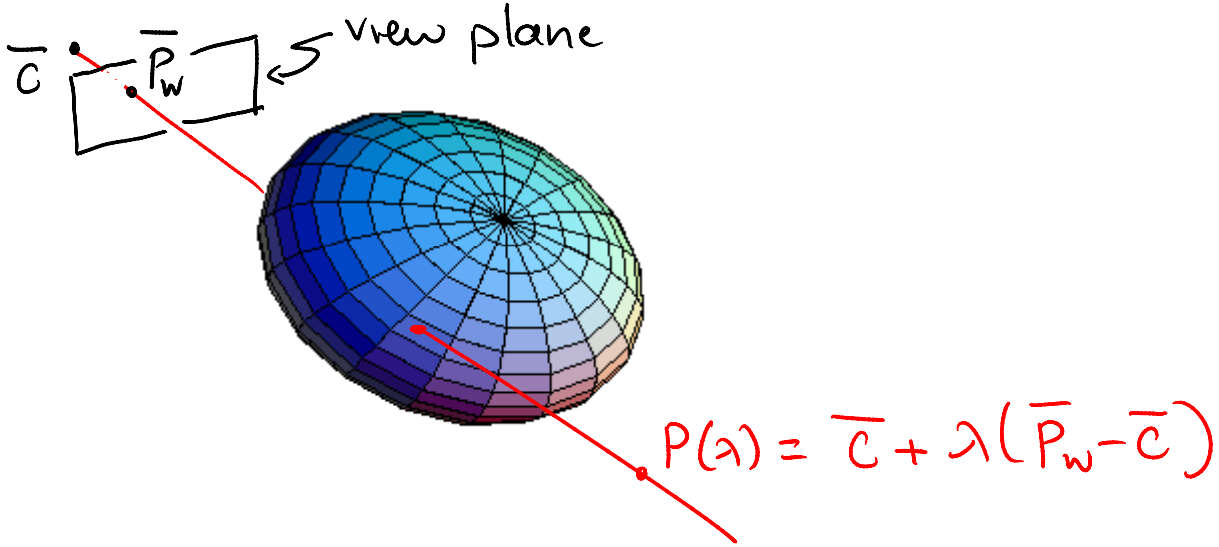
$$P(\lambda)^T \begin{bmatrix} A & D & E & G \\ D & B & F & H \\ E & F & C & I \\ G & H & I & J \end{bmatrix} P(\lambda) = 0$$

the only unknown is λ

expressed in homogeneous 3D coords

for a given quadric, this matrix is known

Computing Ray-Quadric Intersections: 3 Cases



$\Delta > 0$
$\Rightarrow$ 2 "hits"

$\Delta < 0$
$\Rightarrow$ 0 "hits"

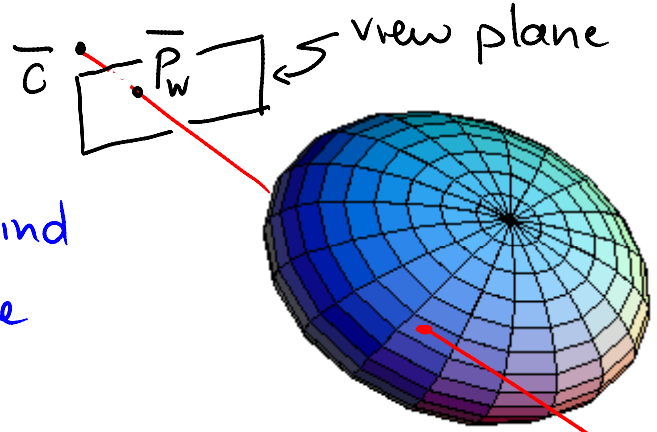
$\Delta = 0$
$\Rightarrow$ 1 "hit"

after expanding, we have a quadratic equation in terms of λ:

$$\alpha\lambda^2 + \beta\lambda + \gamma = 0$$

solution is $\lambda = \frac{-\beta \pm \sqrt{\Delta}}{2\alpha}$, $\Delta = \beta^2 - 4\alpha\gamma$

Ray-Quadric Intersections: Sub-cases for $\Delta>0$



$\lambda_1, \lambda_2 < 0$

$\Rightarrow$ hits are behind the view plane

$\lambda_1 > 0, \lambda_2 > 0$

$\Rightarrow$ 2 valid hits, smallest λ gives intersection closest to camera

$\lambda_1 > 0, \lambda_2 < 0$

$\Rightarrow P(\lambda_1)$ is a valid hit

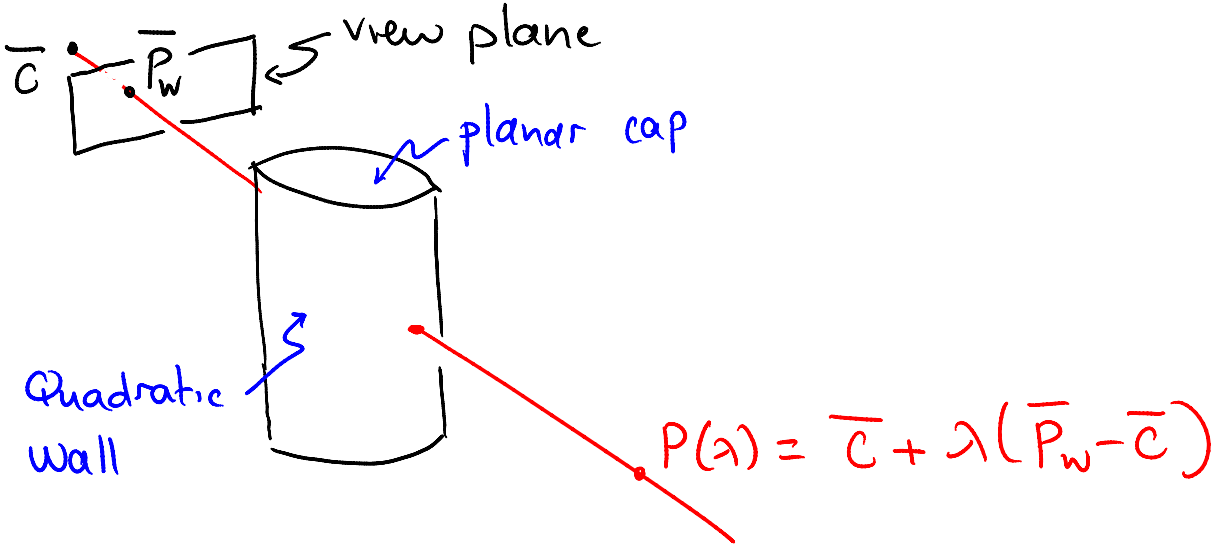
$$P(\lambda) = \bar{C} + \lambda(\bar{P}_w - \bar{C})$$

after expanding, we have a quadratic equation in terms of λ:

$$\alpha\lambda^2 + \beta\lambda + \gamma = 0$$

solution is $\lambda = \dfrac{-\beta \pm \sqrt{\Delta}}{2\alpha}$, $\Delta = \beta^2 - 4\alpha\gamma$

Intersecting Rays & Composite Objects



- When an object is bounded by multiple parametric surfaces we must test for intersection with each of the components
- Example: Cylinder = "Quadratic wall" + 2 planar "caps"

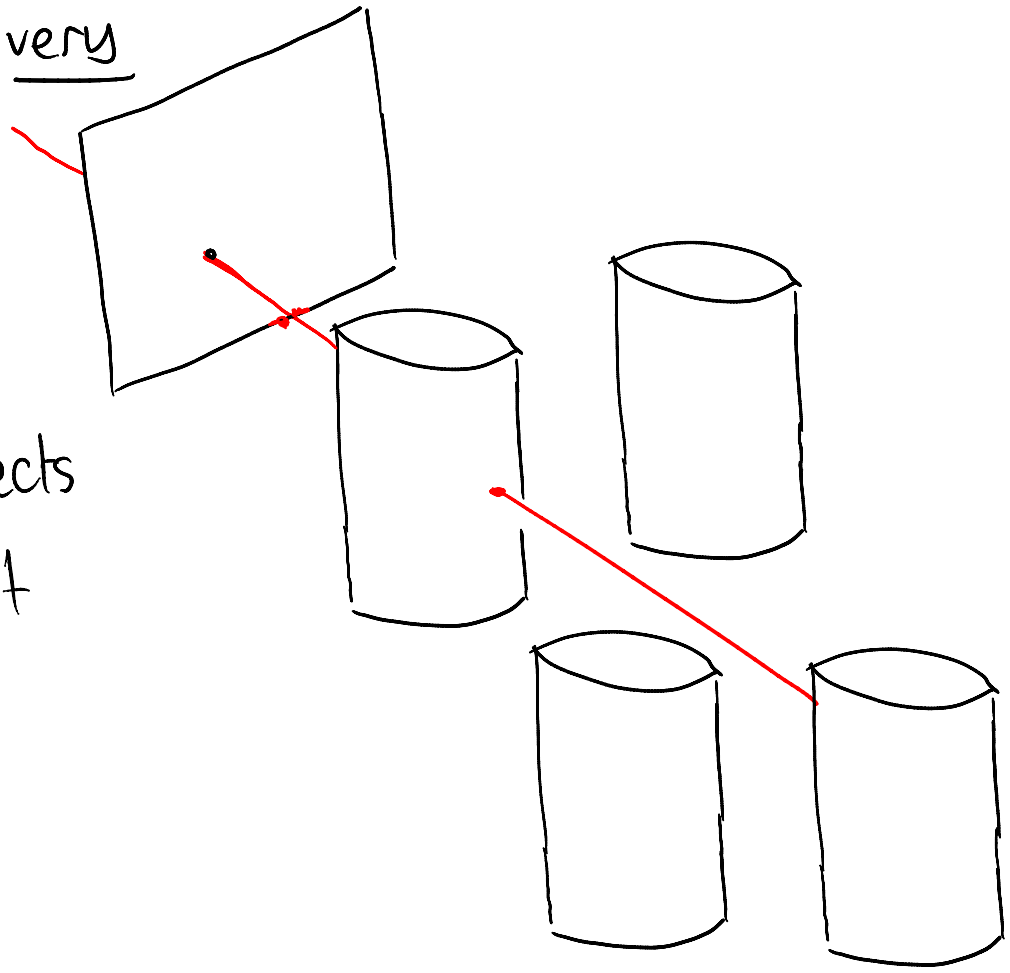
  Cone = "Quadratic wall" + 1 planar base

⇒ See Leonid Sigal's slides for more details

Ray Intersection: Efficiency Considerations

Intersection tests can be very expensive!

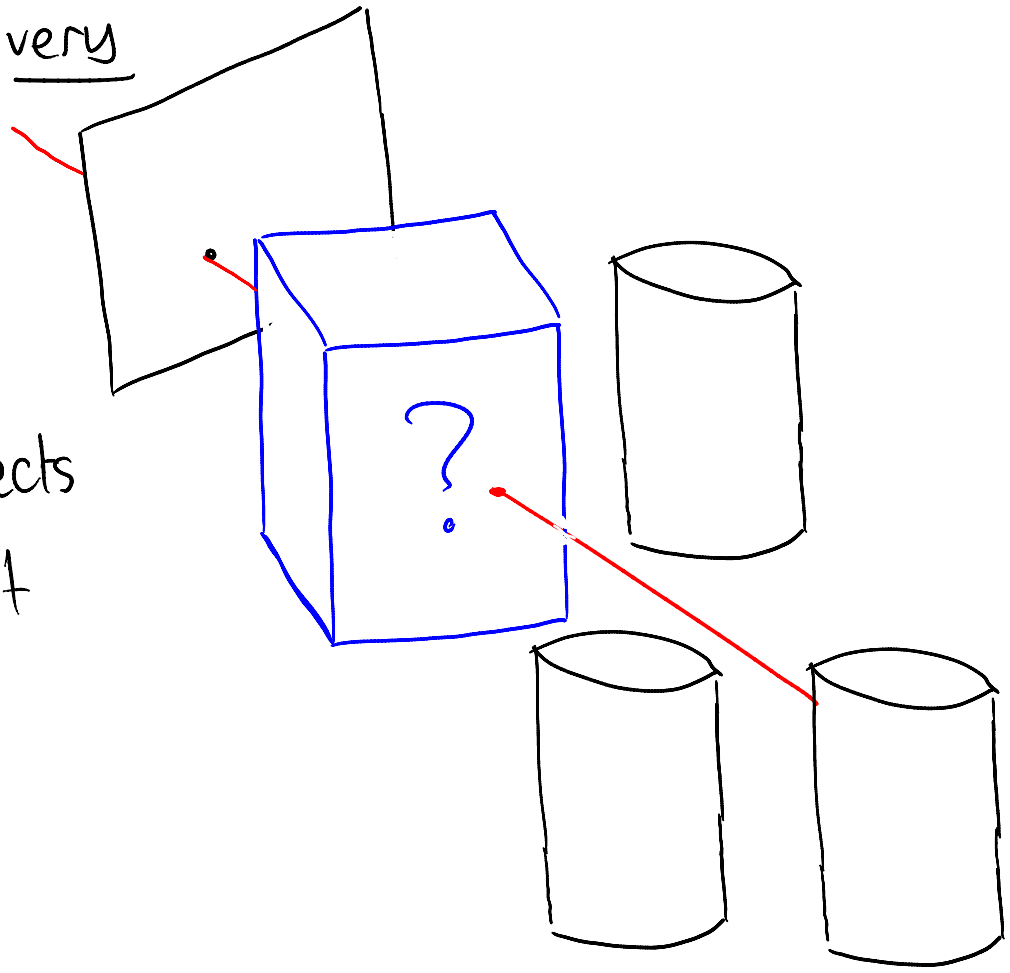
⇒ Use data structures to avoid testing intersection with objects that clearly do not intersect



Ray Intersection: Efficiency Considerations

Intersection tests can be very expensive!

⇒ Use data structures to avoid testing intersection with objects that clearly do not intersect



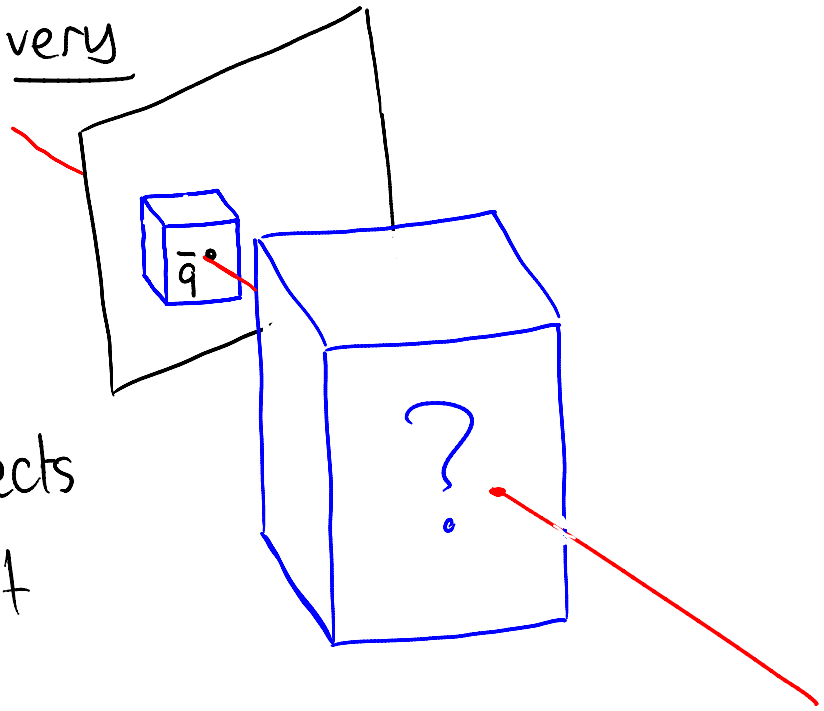
Examples:

- test intersection with objects bounding volume first, test ray-object intersection only if ray intersects volume
- apply this idea hierarchically, for part-based objects

Ray Intersection: Efficiency Considerations

Intersection tests can be very expensive!

⇒ Use data structures to avoid testing intersection with objects that clearly do not intersect



Examples:

- Image-space intersections: instead of intersecting ray & bounding volume, project volume & check whether pixel $\bar{q}$ falls inside that projection

Topic 12:

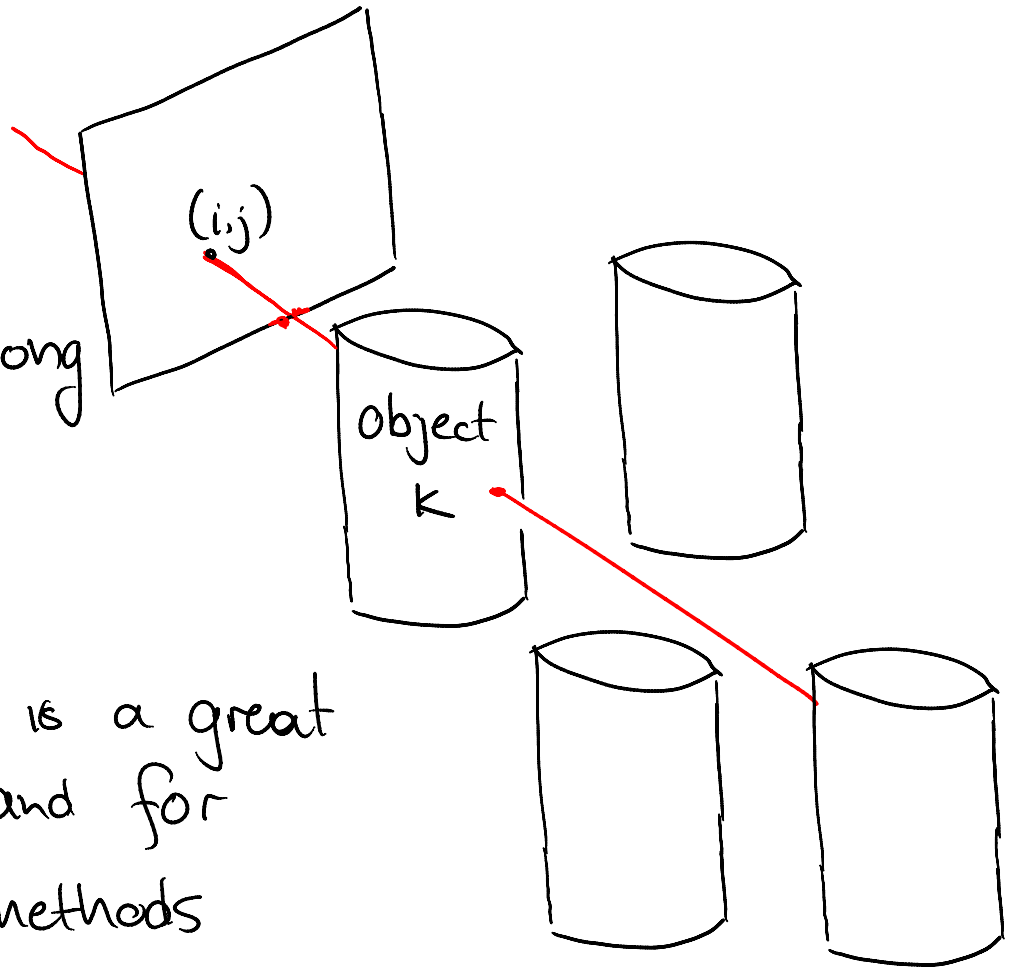
Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

The Scene Signature

Definition:

An image S where $S(i,j) = k$ if object k is the first object along ray through (i,j)

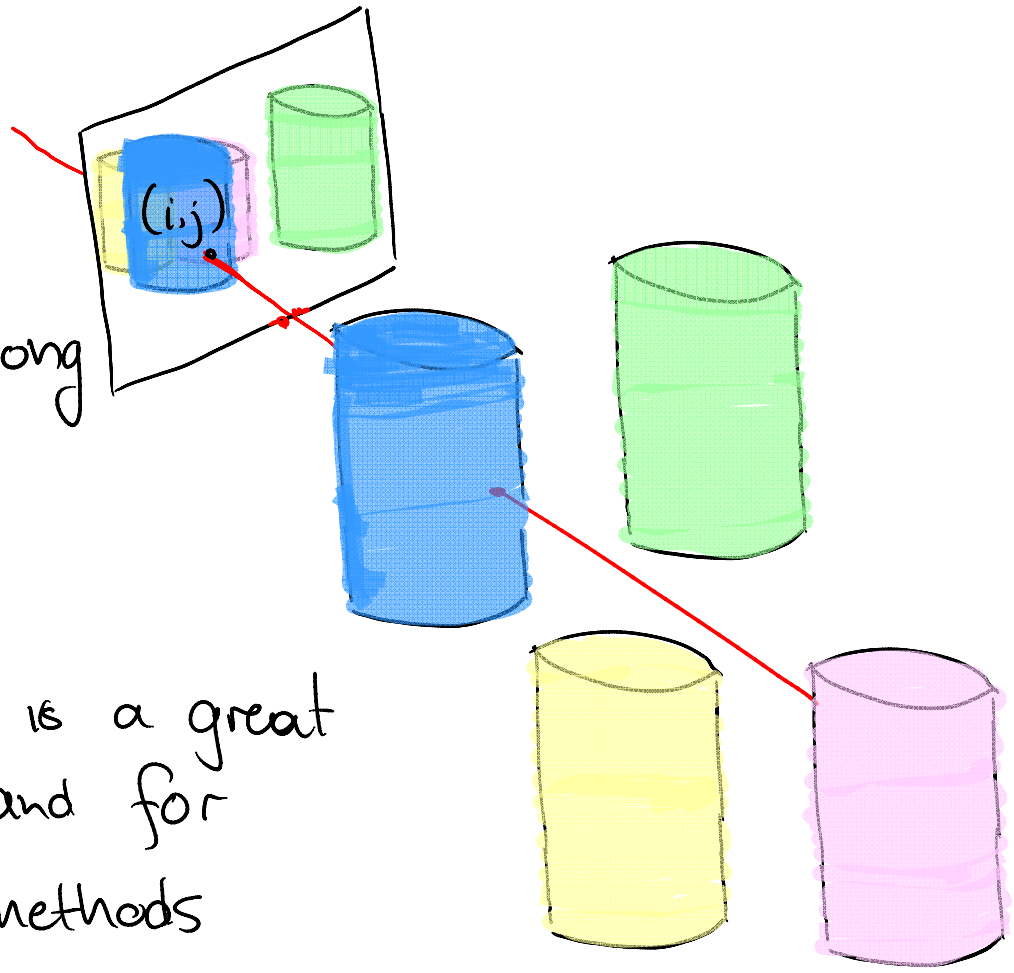


* The scene signature is a great tool for debugging and for testing intersection methods

The Scene Signature

Definition:

An image S where $S(i,j) = k$ if object k is the first object along ray through (i,j)

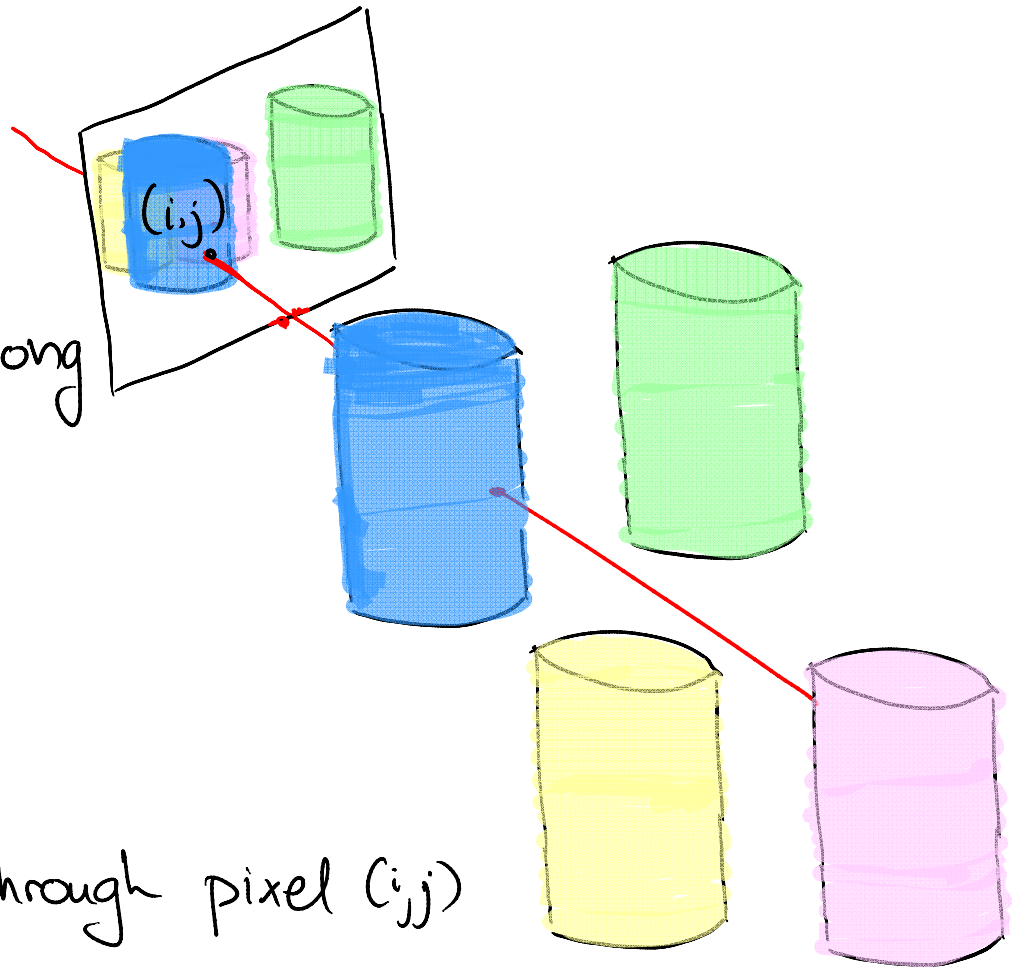


* The scene signature is a great tool for debugging and for testing intersection methods

Computing the Scene Signature

Definition:

An image S where $S(i,j) = k$ if object k is the first object along ray through (i,j)



Algorithm pseudocode:

```
for i = 0 to Nrows-1
    for j=0 to Ncols-1
        construct ray through pixel (i,j)
        λ_{i,j} = ∞
        for k=0 to Nobjects
            λ* = closest intersection of ray with object k
            if λ* > 0 and λ* < λ_{i,j}, set λ_{i,j} = λ*, S(i,j)=k
```

Computational Issues in Basic Ray Tracing

Basic loop:

for each pixel $\bar{q}$

① cast ray r through $\bar{q}$

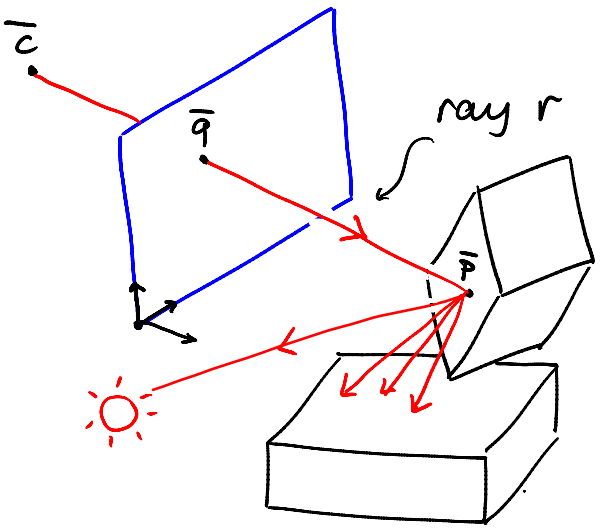
② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

③ estimate amount of light reaching $\bar{p}$

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r

a. Compute surface normal at $\bar{p}$

b. Apply local shading model



Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

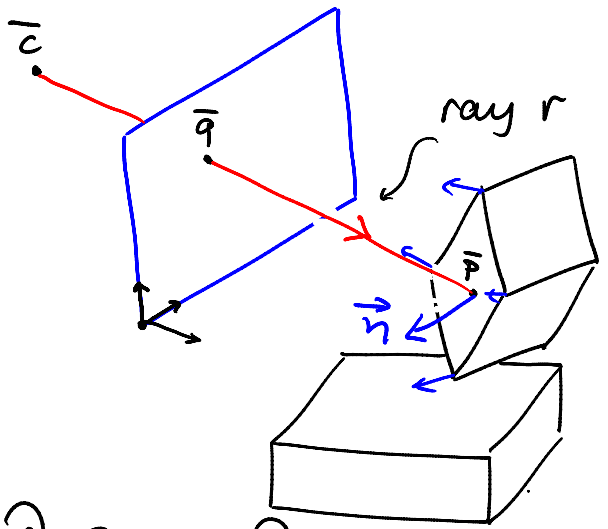
Computing the Normal at a Hit Point

Option #1:

- Smoothly interpolate normal from vertices or adjacent faces (eg. using the linear interpolation technique covered with Phong + scan conversion)

Option #2:

- For parametric shapes, normal can be evaluated directly at $\bar{p}$



Implicit form

$$\vec{n}(\bar{p}) = \frac{\nabla f(\bar{p})}{\| \nabla f(\bar{p}) \|}$$

Explicit form

$\vec{n}(p)$ = unit vector along $\frac{\partial}{\partial \alpha} S(\alpha,\beta) \times \frac{\partial}{\partial \beta} S(\alpha,\beta)$

Computing the Normal at a Hit Point

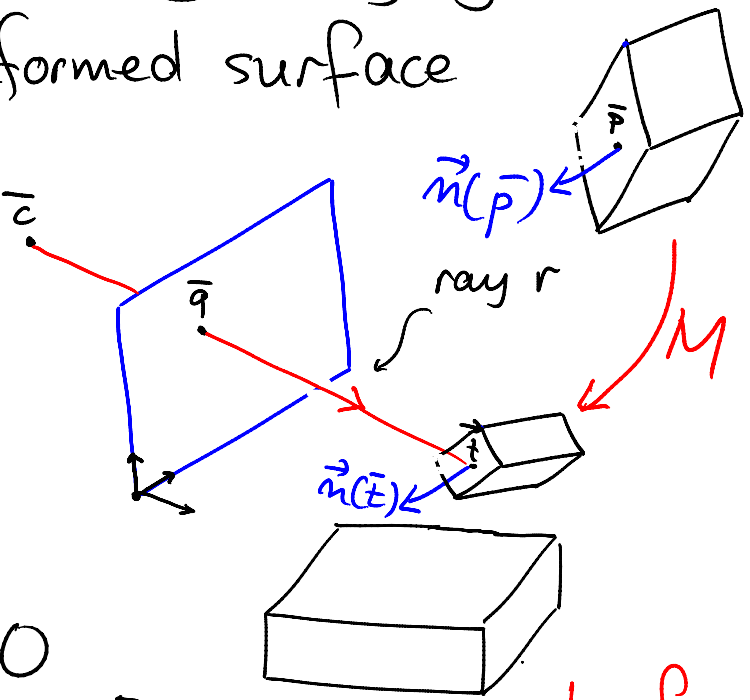
Option #3 (affinely-deformed shapes)

- Let $f(\bar{p}) = 0$ be an implicit surface
- Let M be a 4x4 affine transformation matrix
- Suppose we deform the surface by applying M to it
- Point $\bar{t}$ will be on the deformed surface if there is a $\bar{p}$ such that $\bar{t} = M\bar{p}$ ← expressed in homogeneous coords

$$\Leftrightarrow \quad \bar{p} = M^{-1}\bar{t}$$

$$\Leftrightarrow \text{ implicit eq is } \quad F(\bar{t}) = f(M^{-1}\bar{p}) = 0$$

$$\Leftrightarrow \quad \vec{n}(\bar{t}) = (M^{-1})^T \vec{n}(\bar{p}) \,/\, \|(M^{-1})^T \vec{n}(\bar{p})\|$$ ← see notes for complete proof



Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Evaluating the Shading Model

Use a two-component model

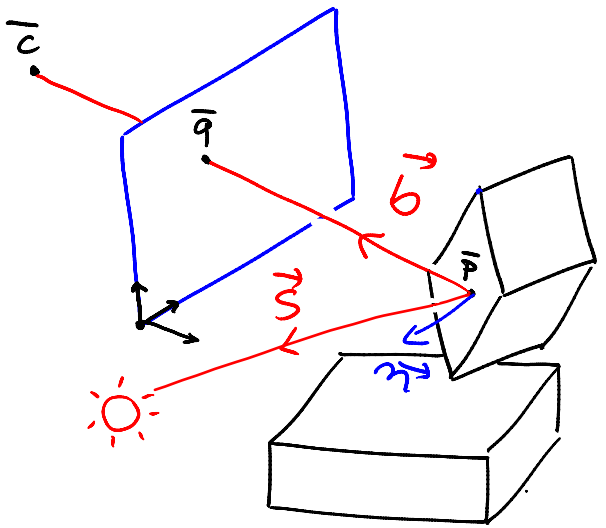
$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

r_s ← specular reflection coeff

$I(\bar{q})$: intensity at pixel $\bar{q}$

$L(\vec{b}, \vec{n}, \vec{s})$: local shading model at $\bar{p}$

$G(\bar{p})$: global shading component at $\bar{p}$



Evaluating the Shading Model

Use a two-component model

$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

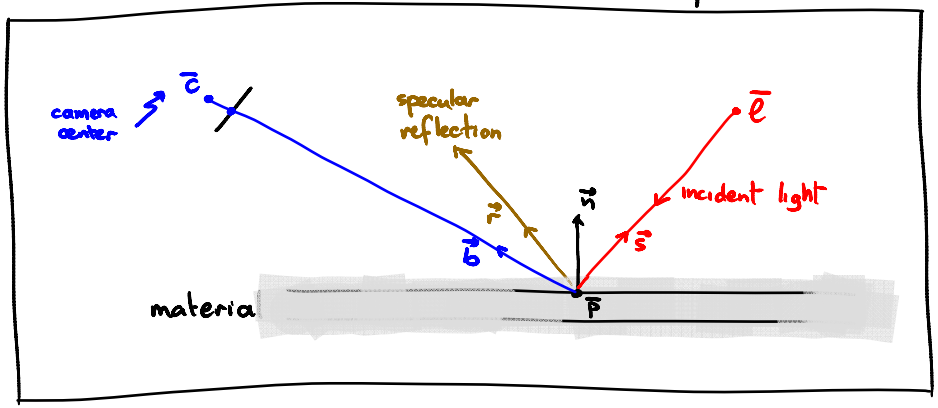
Phong model

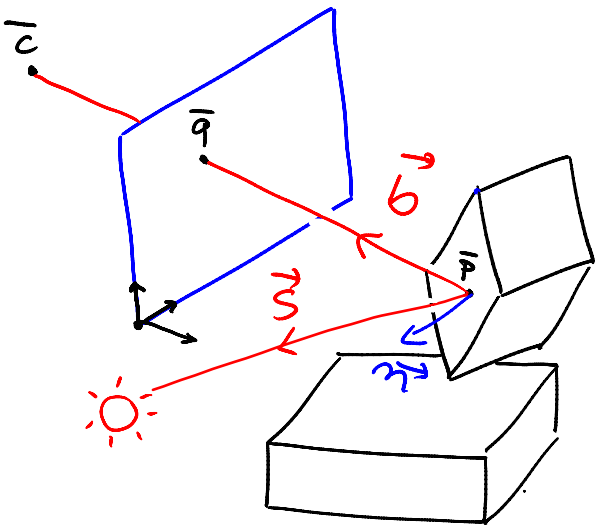
$$r_a I_\alpha + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^\alpha$$

ambient — diffuse — specular

Computed after ray spawning

close-up "view" of point $\bar{p}$





Evaluating the Shading Model: Using Textures

Use a two-component model

$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

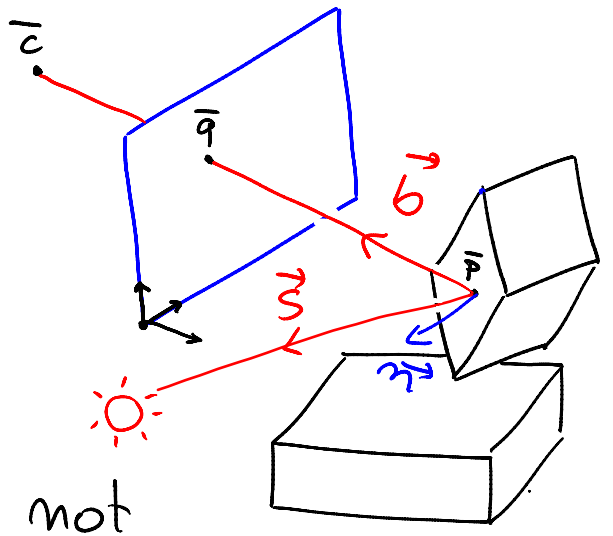
$$r_a I_a + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^{\alpha}$$

ambient | diffuse | specular

Computed after ray spawning

Texture can be used to modulate r_a and r_d:

- need to compute $\bar{p}$'s texture coordmates
- unlike scan-conversion, we compute $\bar{p}$'s texture coordmates by linear interpolation on the polygon plane, not in image space (⇐ no distortion artifacts)



Computational Issues in Basic Ray Tracing

Basic loop:

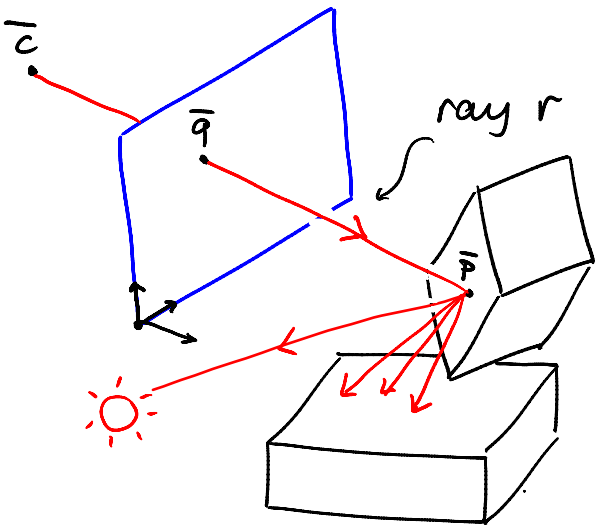
for each pixel $\bar{q}$

① cast ray r through $\bar{q}$

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

③ estimate amount of light reaching $\bar{p}$

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r



Computational Issues in Basic Ray Tracing

Basic loop:

for each pixel $\bar{q}$

① cast ray r through $\bar{q}$

② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

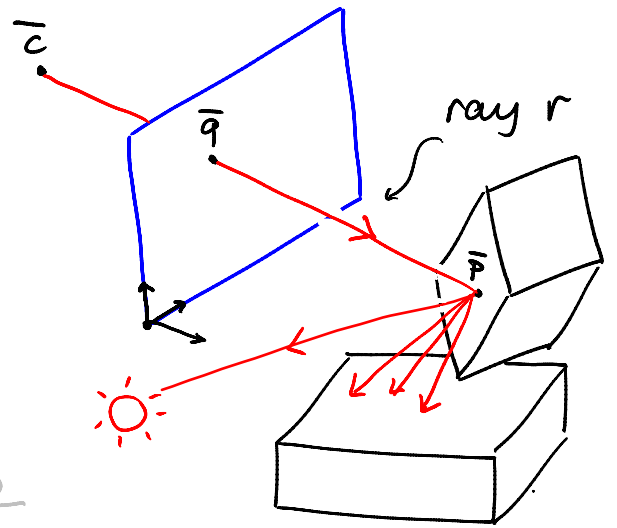
③ estimate amount of light reaching $\bar{p}$

a. "spawn" rays $r_1, r_2, \ldots, r_k$ from $\bar{p}$ in various directions

b. if ray r_i hits a light source, estimate light travelling along r_i and stop

c. else apply loop recursively to ray r_i

④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r



Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Whitted Ray Tracing

Basic idea:

- Spawn only <u>one</u> ray
- Ray is along ideal specular direction

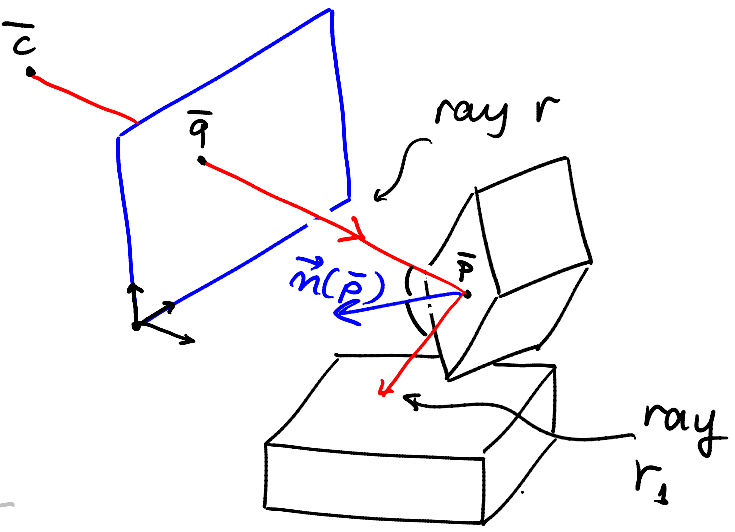
---

③ estimate amount of light reaching $\bar{P}$

a. "spawn" rays ~~$r_1, r_2, \ldots, r_k$~~ r_1 from $\bar{P}$ in ~~various~~ specular direction ~~directions~~ only

b. if ray r_i hits a light source, estimate light travelling along r_i and <u>stop</u>

c. else apply loop recursively to ray r_i



Whitted Ray Tracing

Motivation:

- Computationally efficient (1 spawned ray / bounce)
- Models the most important light path (in terms of "light energy" transfered from r_i to r)

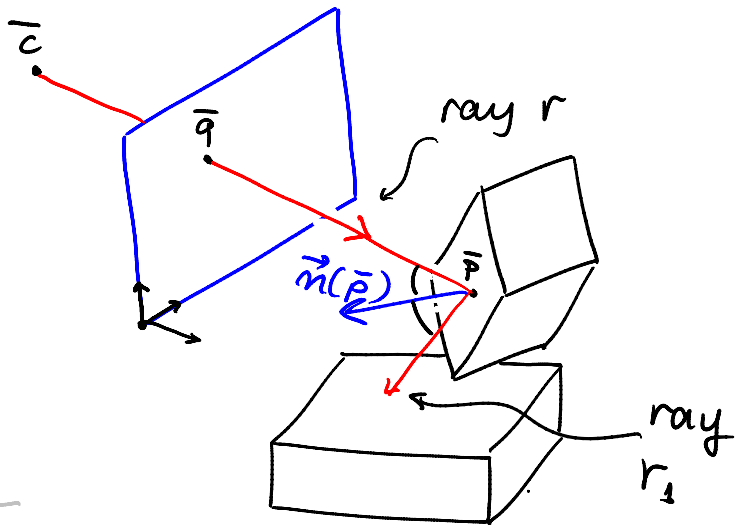
---

③ estimate amount of light reaching $\bar{p}$

a. "spawn" rays ~~$r_1, r_2, \ldots, r_k$~~ r_1 from $\bar{p}$ in ~~various~~ specular direction ~~directions~~ only

b. if ray r_i hits a light source, estimate light travelling along r_i and stop

c. else apply loop recursively to ray r_i



Whitted Ray Tracing: An Example

Use a two-component model

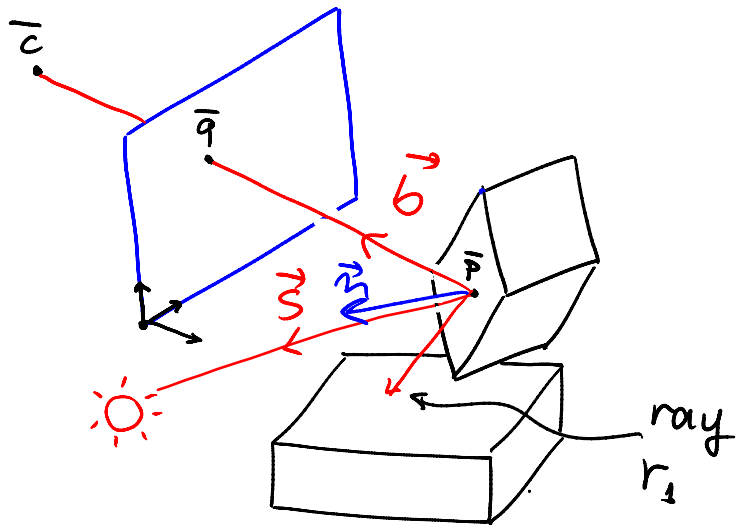
$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

$$r_a I_\alpha + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^\alpha$$

ambient — diffuse — specular

Global specular term



Whitted Ray Tracing: An Example

Use a two-component model

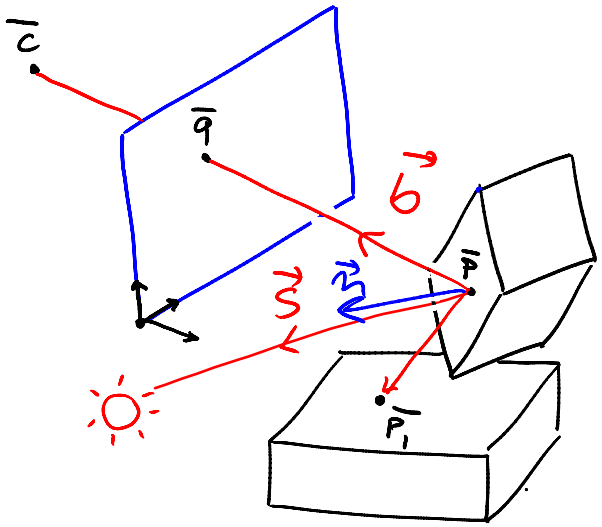
$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

$$r_a I_\alpha + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^\alpha$$

ambient — diffuse — specular

Global specular term



Whitted Ray Tracing: An Example

Use a two-component model

$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

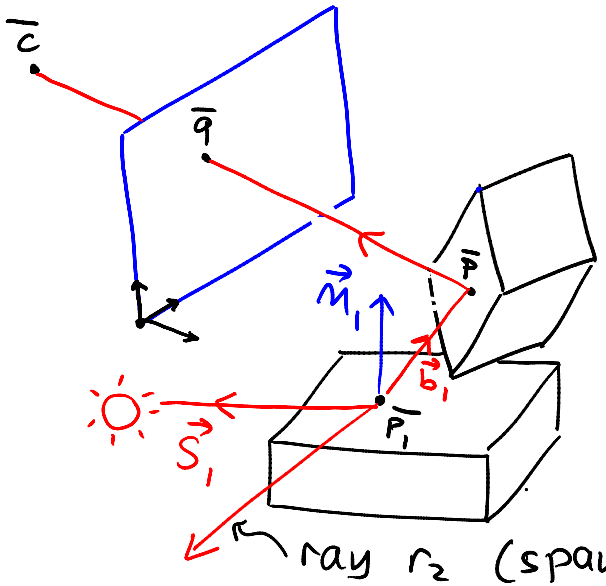
Phong model

$$r_a I_\alpha + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^\alpha$$

ambient — diffuse — specular

Global specular term

$$L(\vec{b}_1, \vec{n}_1, \vec{s}_1) + G(\bar{p}_1) r_s$$



ray r_2 (spawned from $\bar{p}_1$)

Whitted Ray Tracing: An Example

Use a two-component model

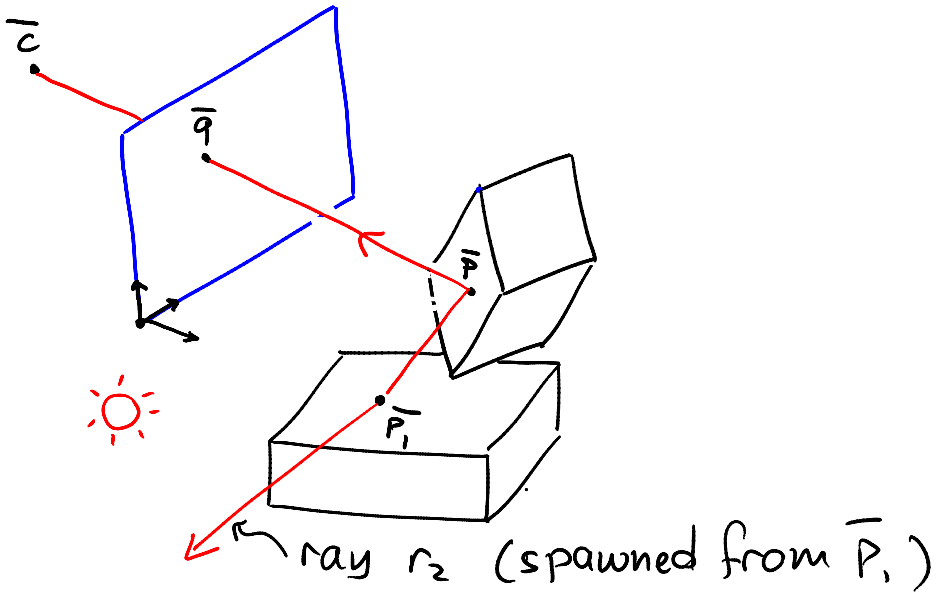
$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

$$r_a I_\alpha + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^\alpha$$

ambient — diffuse — specular

Global specular term



$$L(\vec{b}_1, \vec{n}_1, \vec{s}_1) + G(\bar{p}_1) r_s$$

computed by recursively ray tracing along r_2

Simulating Shadows

Use a two-component model

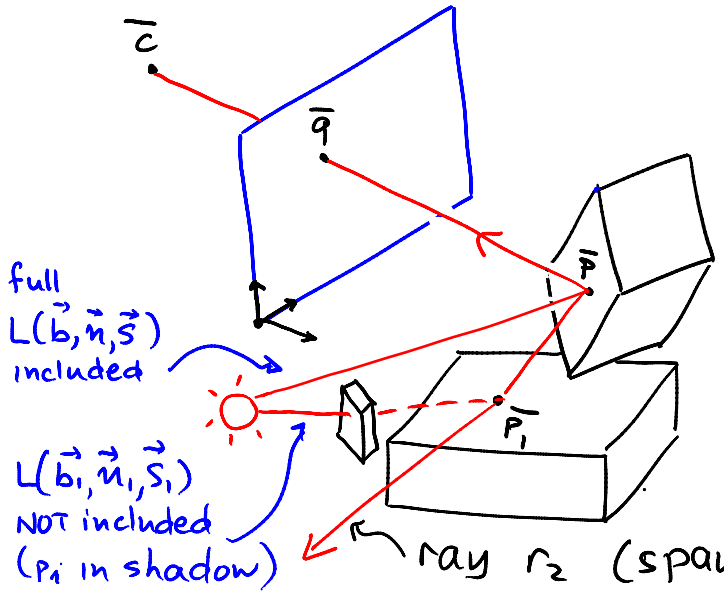
$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

$$r_a I_\alpha + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^\alpha$$

ambient | diffuse | specular

Global specular term



if p_i not visible to light source, include only ambient term

$$L(\vec{b}_1, \vec{n}_1, \vec{s}_1) \quad G(\bar{p}_1) r_s$$

computed by recursively ray tracing along r_2

Simulating Shadows

Use a two-component model

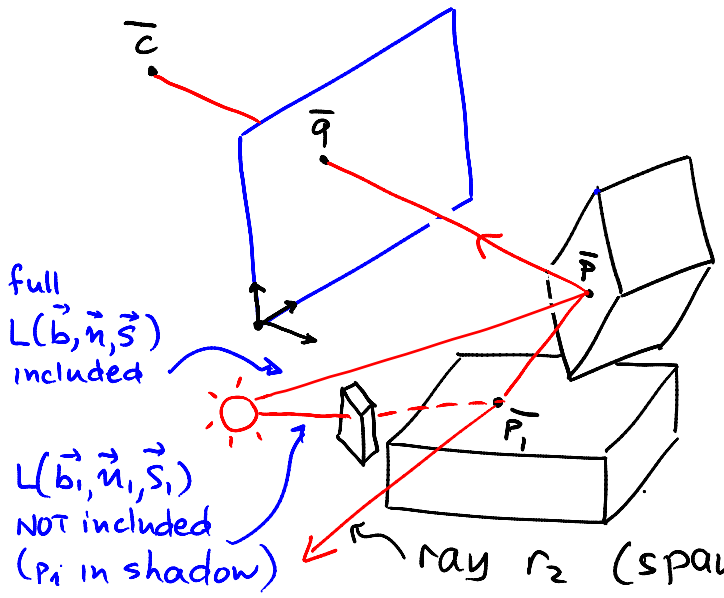
$$I(\bar{q}) = L(\vec{b}, \vec{n}, \vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

$$r_a I_a + r_d I_d \max(0, \vec{n} \cdot \vec{s}) + r_s I_s \max(0, \vec{r} \cdot \vec{b})^{\alpha}$$

ambient — diffuse — specular

Global specular term



if p_i not visible to light source, include only ambient term

$r_a \cdot I_a$

$G(\bar{p}_1) r_s$

computed by recursively ray tracing along r_2

Simulating Shadows

Use a two-component model

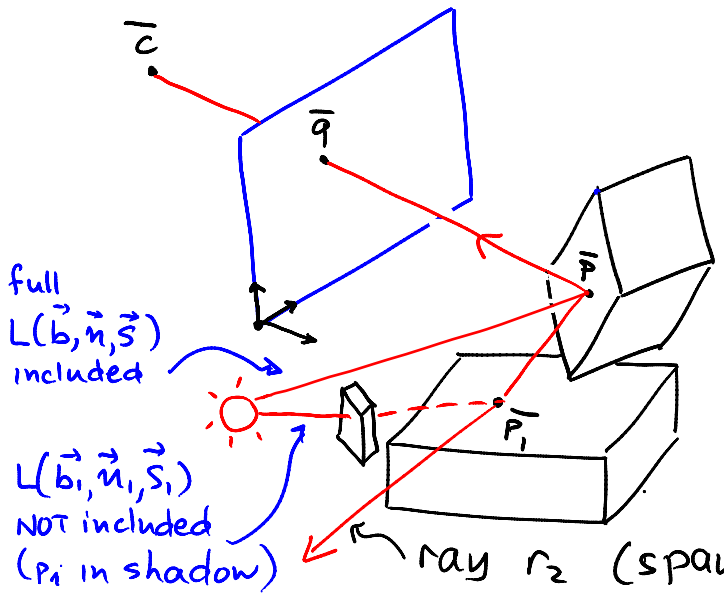
$$I(\bar{q}) = L(\vec{b},\vec{n},\vec{s}) + G(\bar{p}) \cdot r_s$$

Phong model

$$r_a I_a + r_d I_d \max(0, \vec{n}\cdot\vec{s}) + r_s I_s \max(0, \vec{r}\cdot\vec{b})^{\alpha}$$

ambient | diffuse | specular

Global specular term



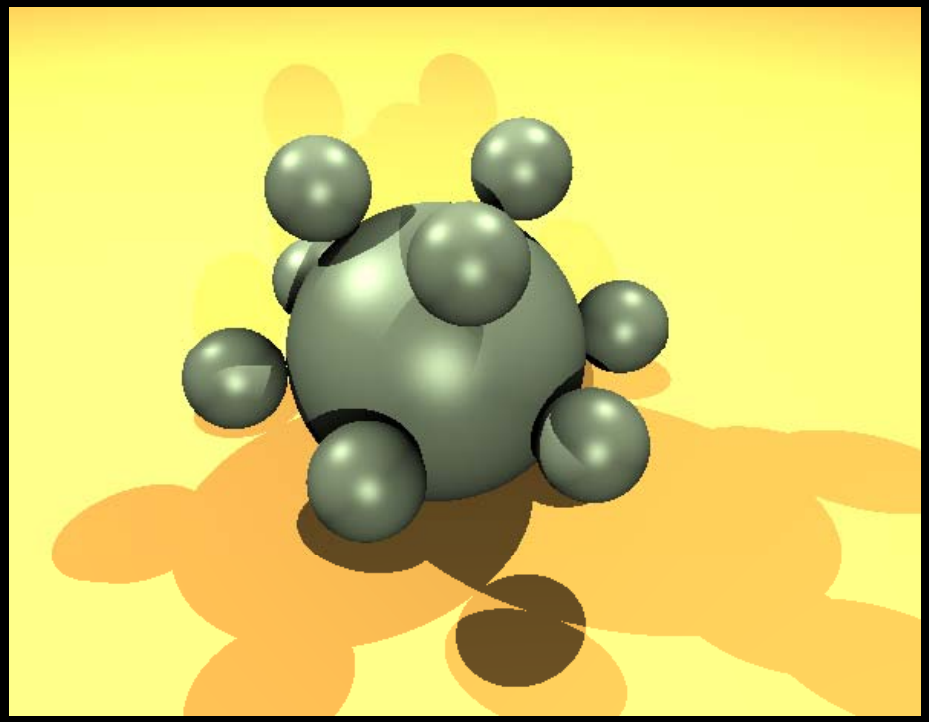
if p_i not visible to light source, include only ambient term

$$r_a \cdot I_a + G(\bar{p}_1) r_s$$

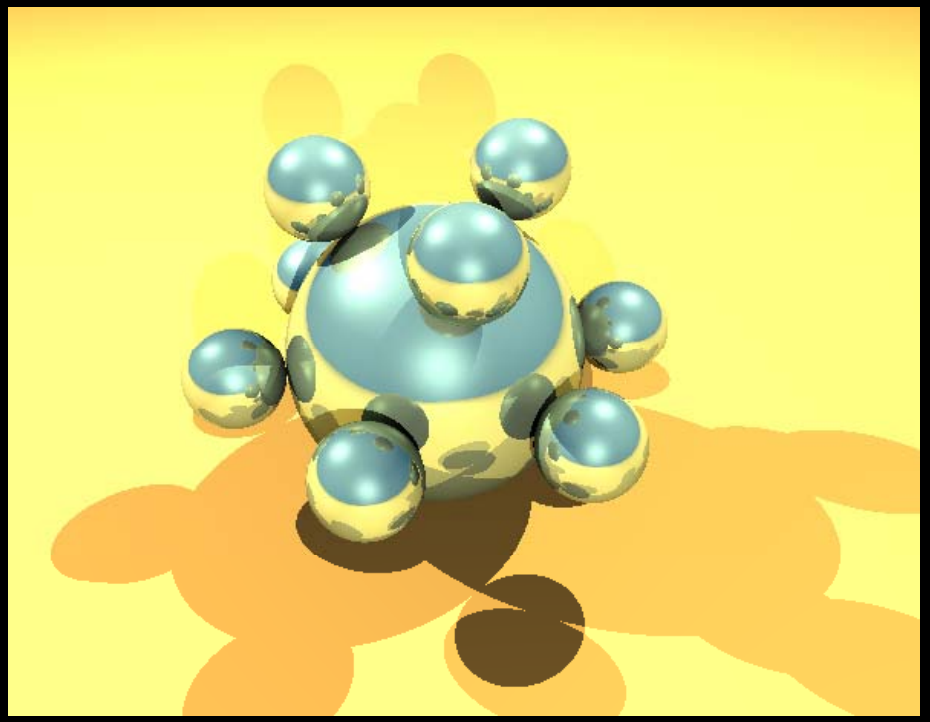
determined by casting a ray from p_i to the (point) light source and testing for object hits between p_i & that source

Non-recursive vs. Recursive Ray Tracing

No recursion (local lighting + shadows)



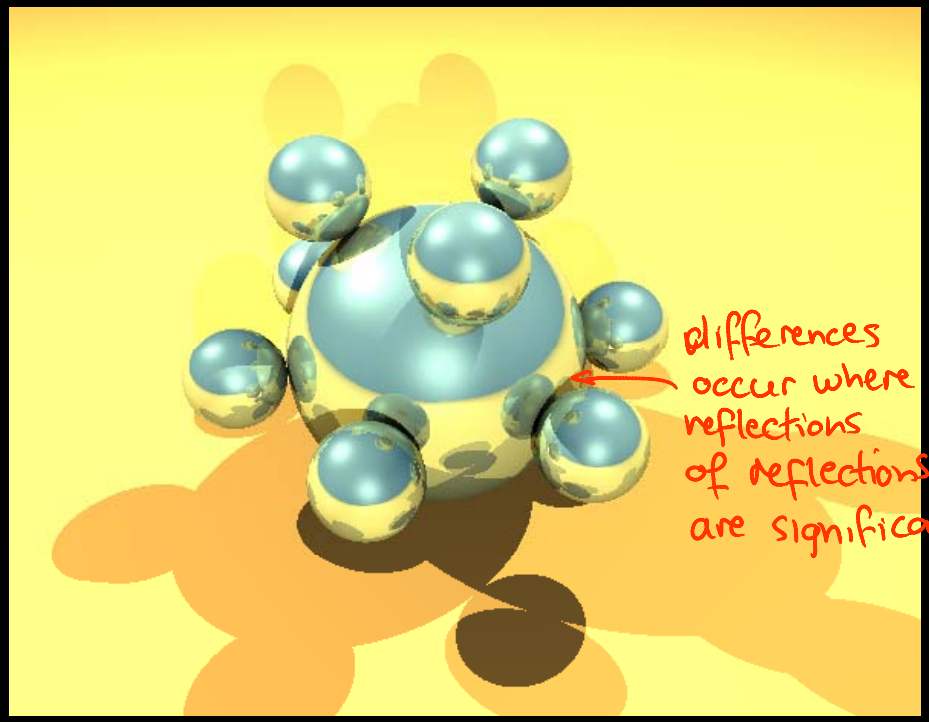
1 Level of recursive spec. reflection



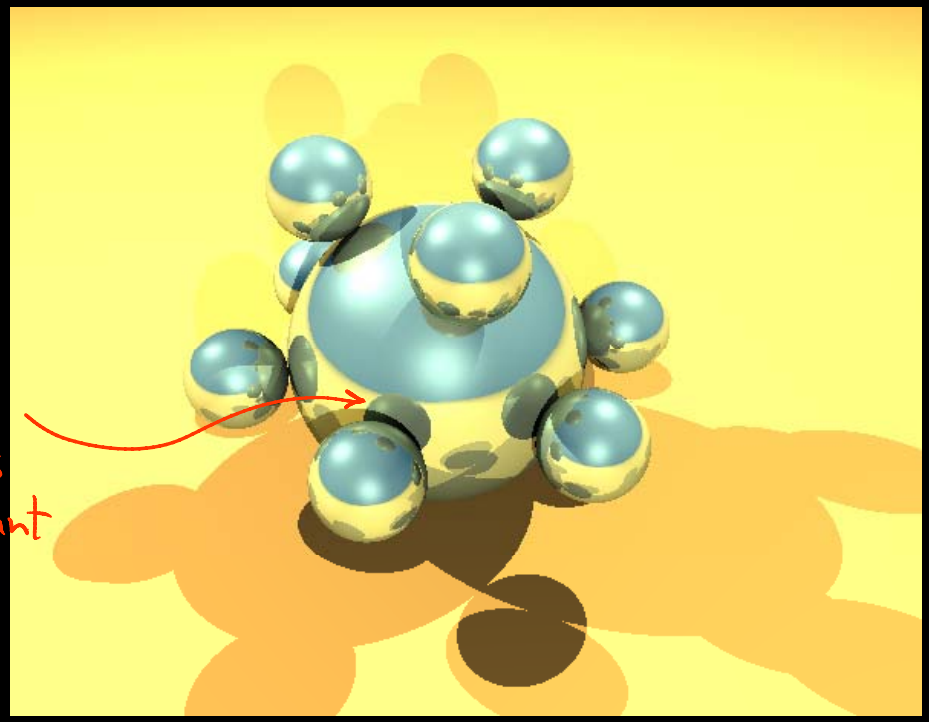
https://agora.cs.uiuc.edu/download/attachments/10454060/RayTracing_suppl.ppt?version=1

Ray tracing in the movies

2 Levels of recursive spec. reflection



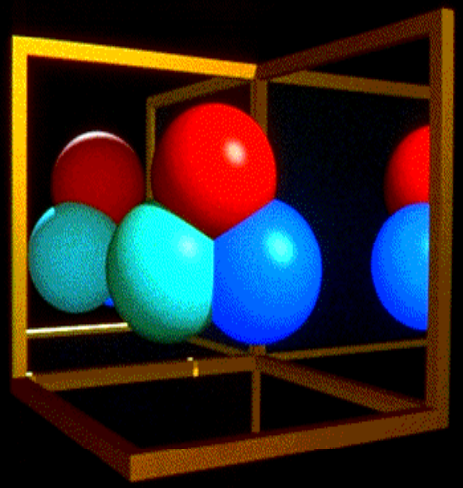
1 Level of recursive spec. reflection



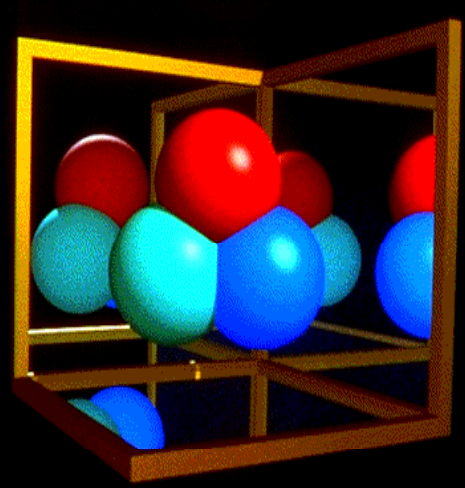
https://agora.cs.uiuc.edu/download/attachments/10454060/RayTracing_suppl.ppt?version=1

Ray tracing in the movies

1 recursive level



2 recursive levels



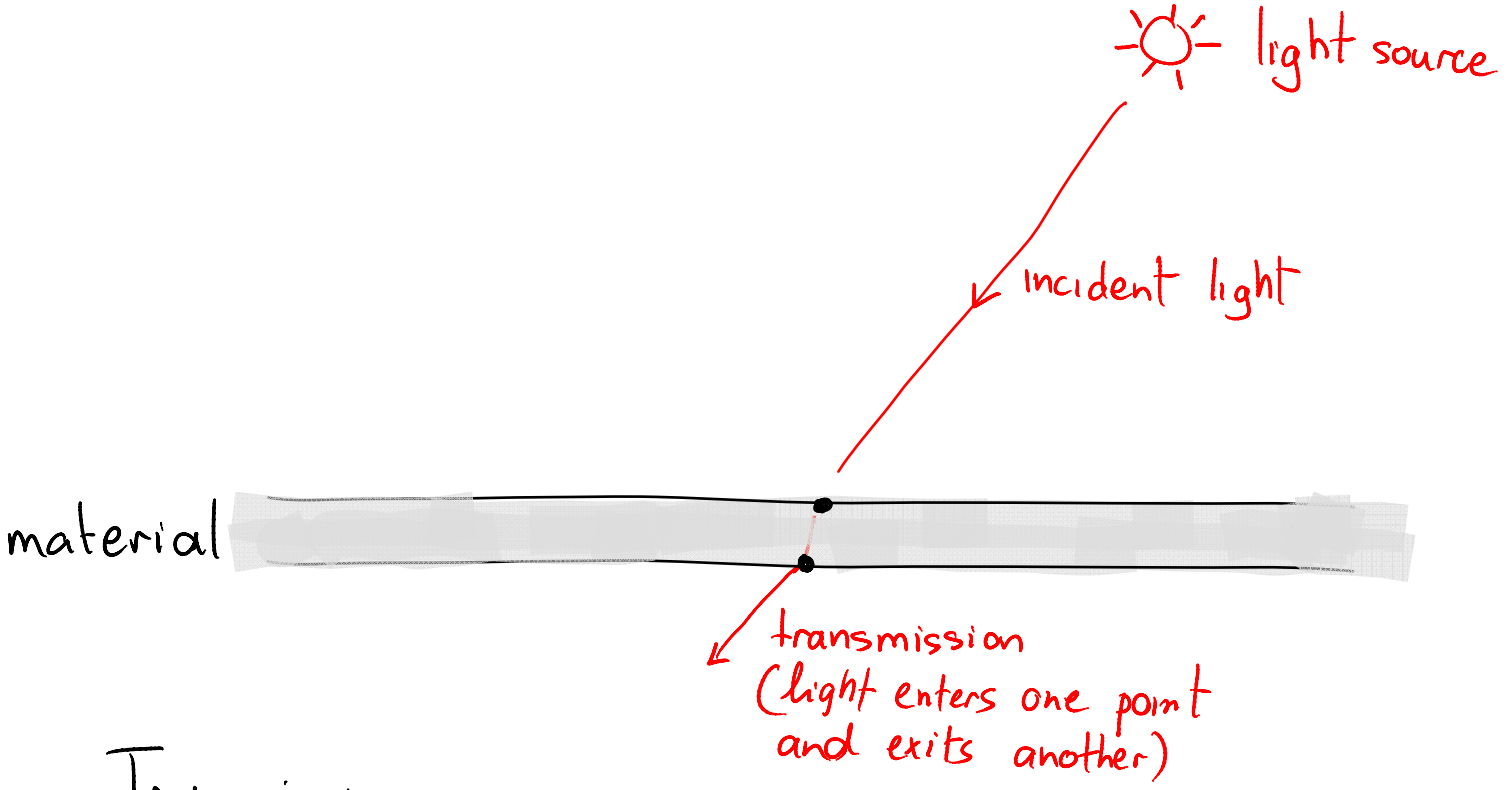
Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction


Guet al, EGSR'07

Modeling Reflection: Transmission

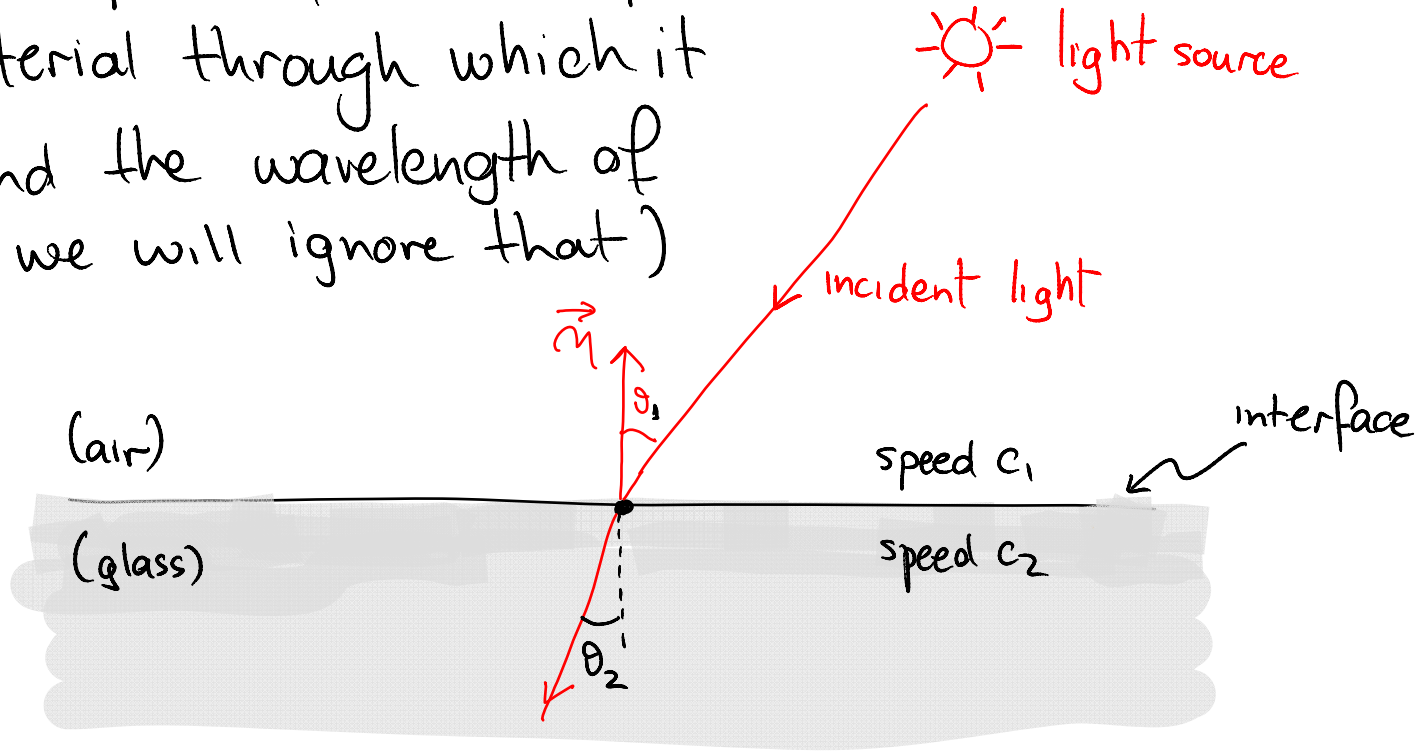


Transmission:

- Caused by materials that are not perfectly opaque
- Examples include glass, water and translucent materials such as skin

Physics of Refraction

Physics: the speed of light depends on the material through which it travels (and the wavelength of light, but we will ignore that)



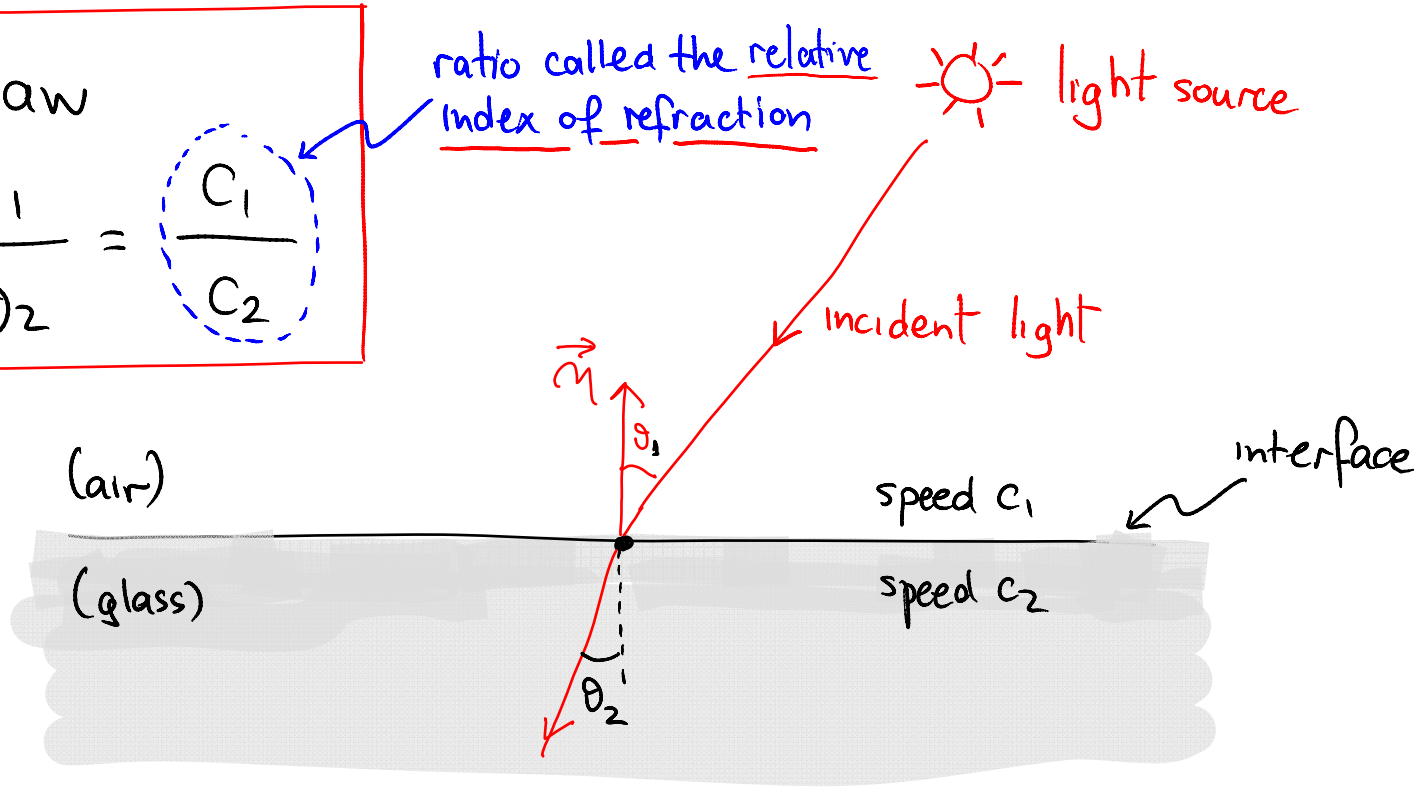
Refraction (bending of rays) occurs when light crosses an interface between two media with different speeds of light

Physics of Refraction

Snell's law

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

ratio called the relative index of refraction

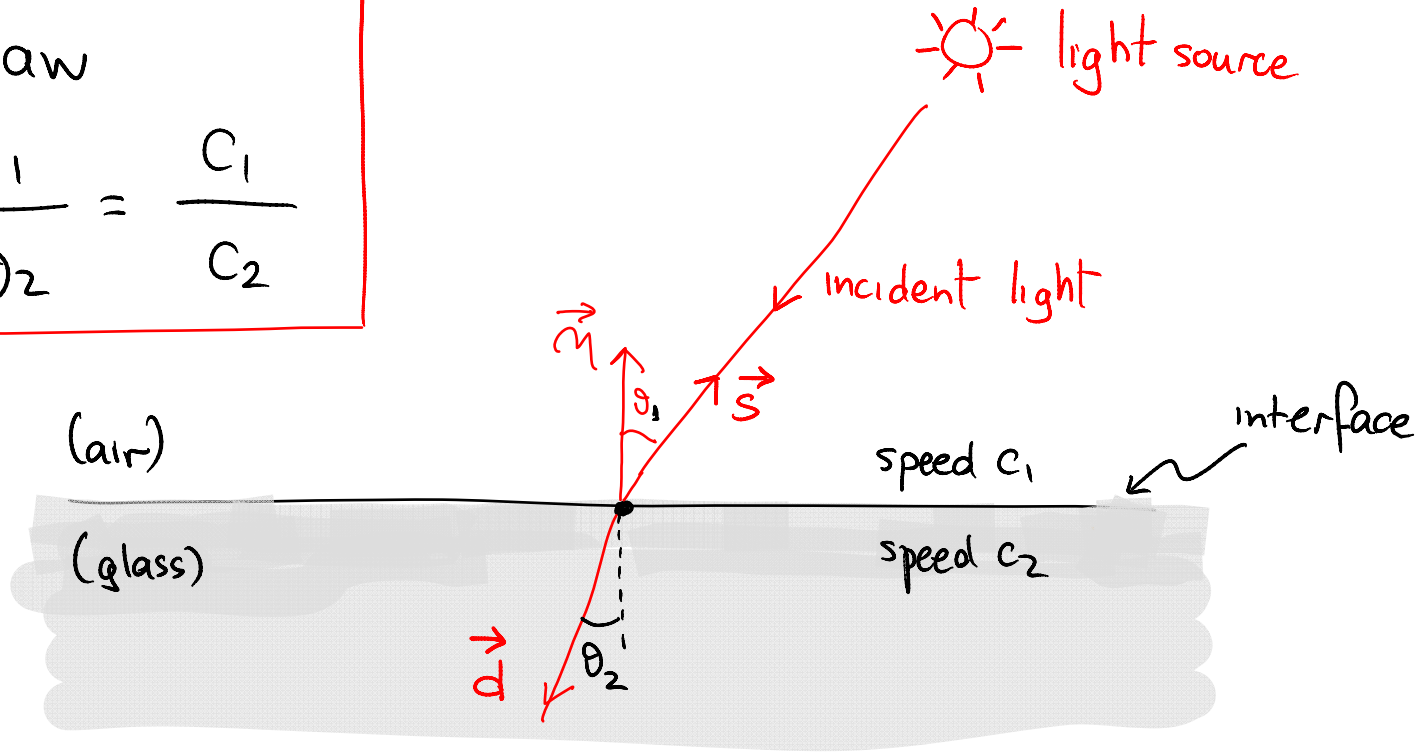


Refraction (bending of rays) occurs when light crosses an interface between two media with different speeds of light

Geometry of Refraction: Transmission Vector

Snell's law

$$\frac{\sin\vartheta_1}{\sin\vartheta_2} = \frac{c_1}{c_2}$$



① Incident ray, outgoing ray & normal always lie on the same plane $\Rightarrow$

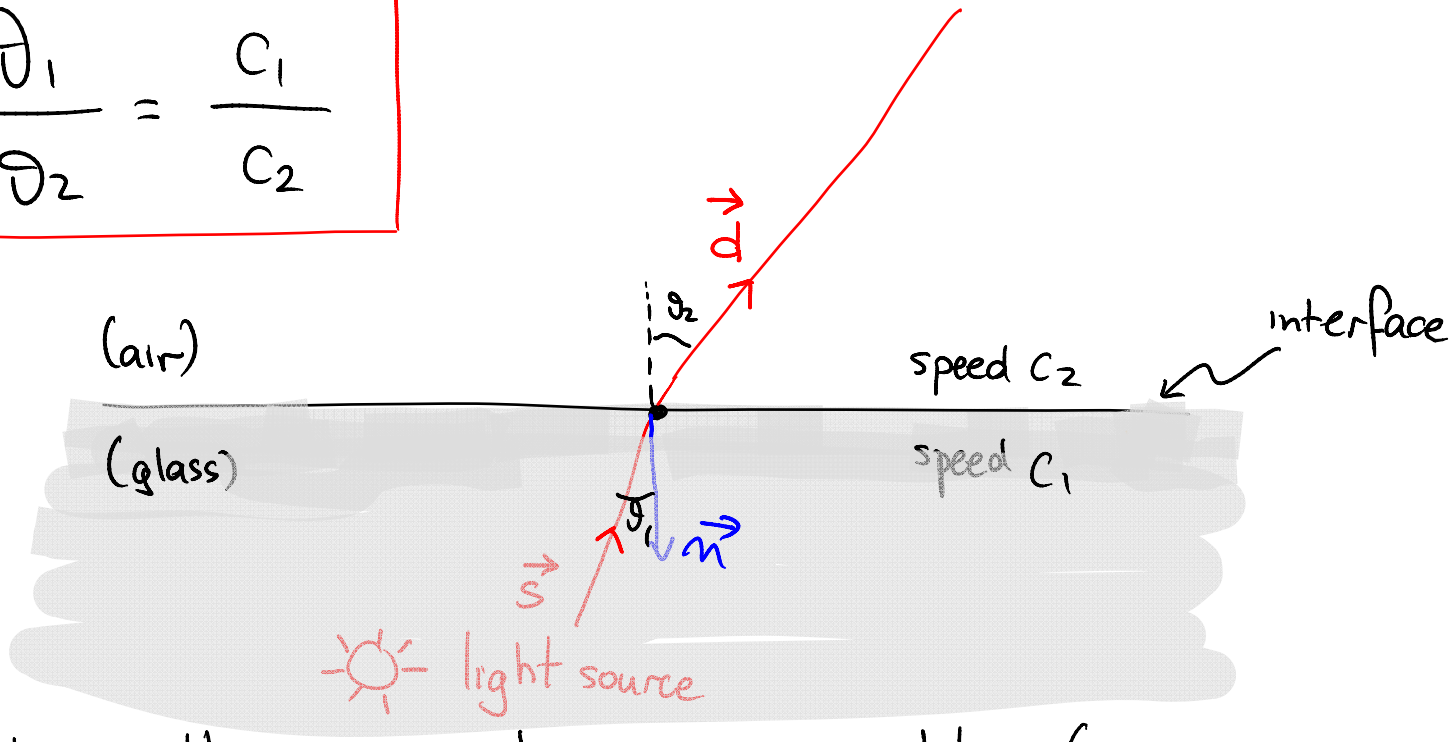
$\vec{d}$ along $-\frac{c_2}{c_1}\vec{s} + \left[\frac{c_2}{c_1}\cos\vartheta_1 - \cos\vartheta_2\right]\vec{n}$

exercise: prove this

Geometry of Refraction: Path Reversibility

Snell's law

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

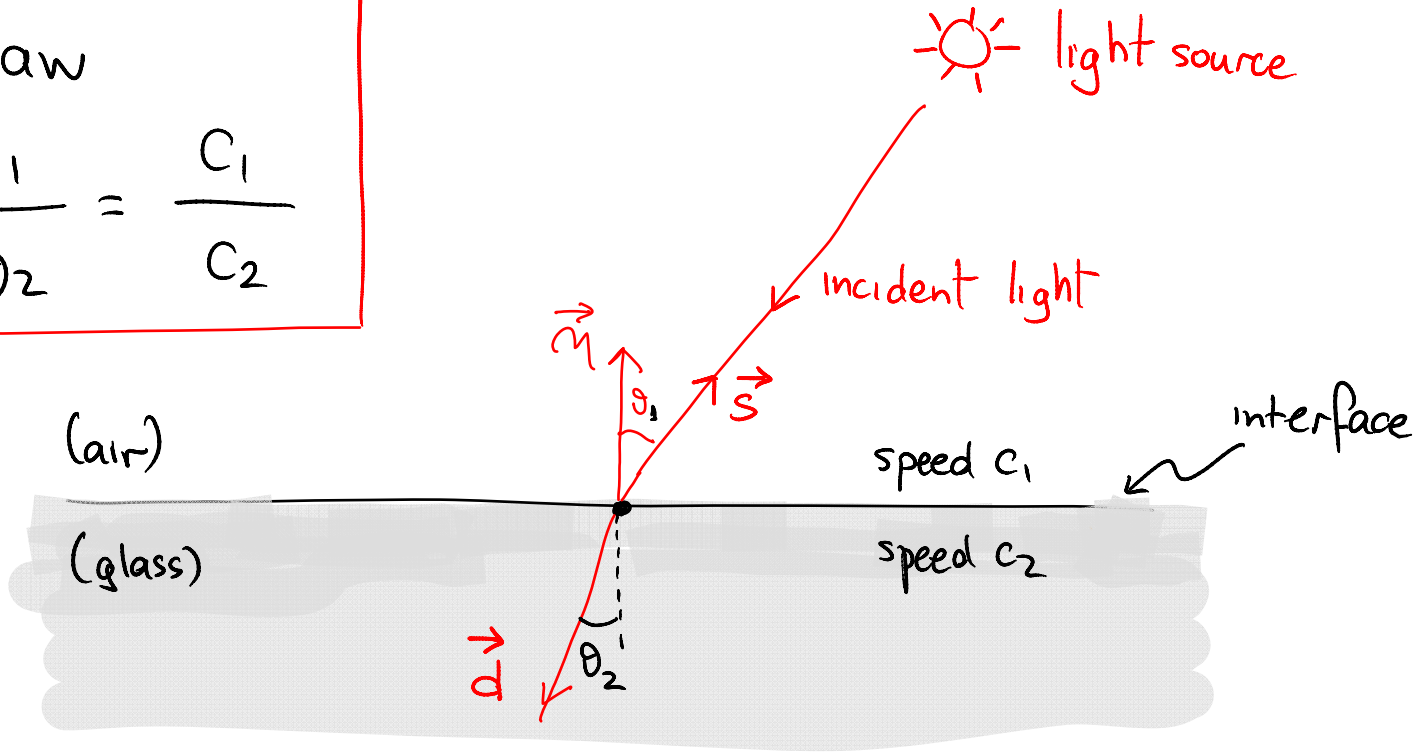


② Light paths are always reversible (i.e. light is transmitted exactly the same way if its direction of travel is reversed)

Geometry of Refraction

Snell's law

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

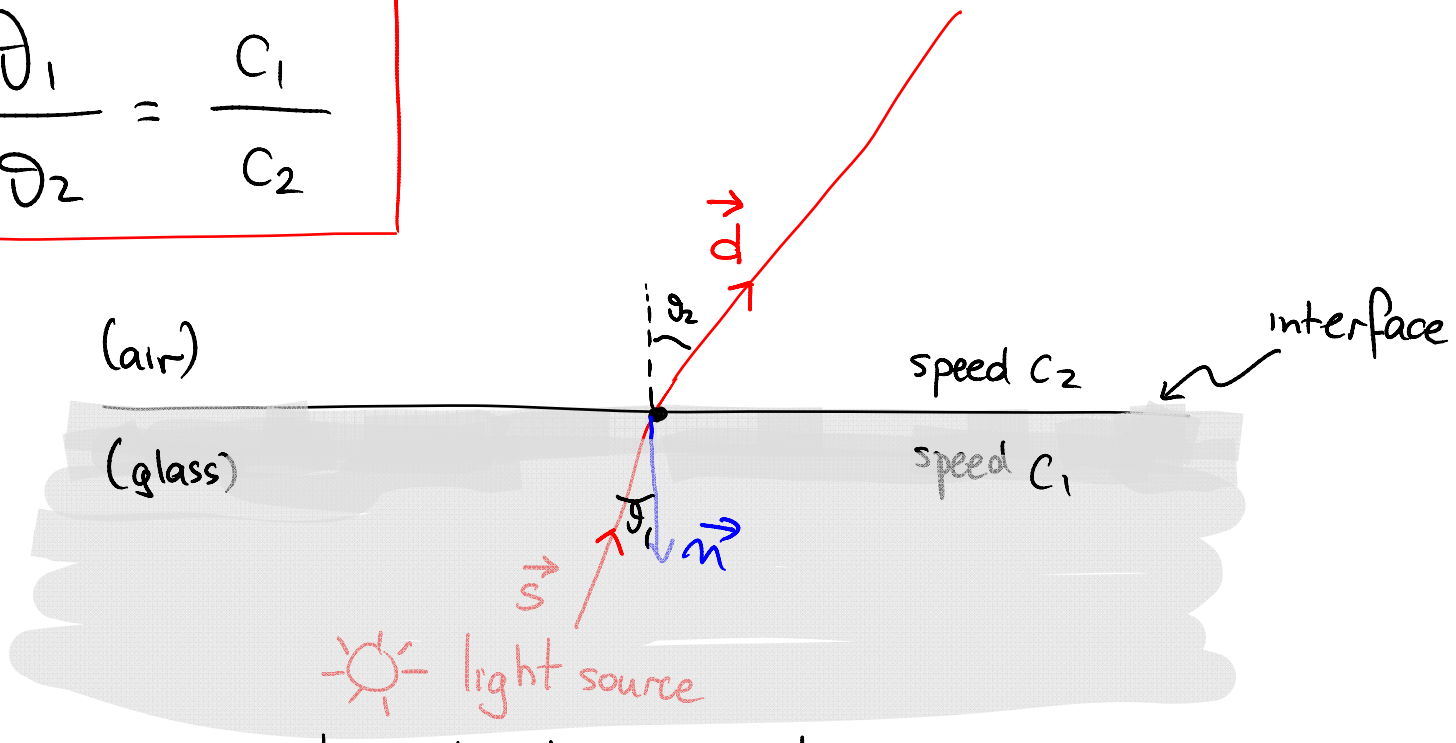


③ If $c_2 < c_1$ light bends toward the normal

Geometry of Refraction

Snell's law

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



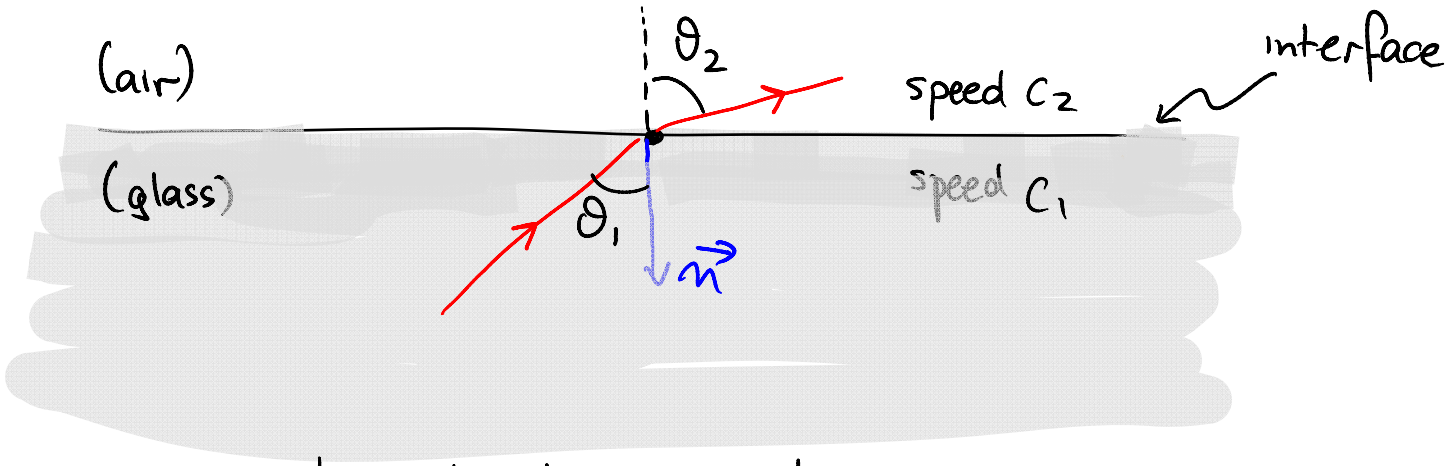
③ If $c_2 < c_1$ light bends toward the normal

If $c_2 > c_1$ light bends away from normal

Geometry of Refraction

Snell's law

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



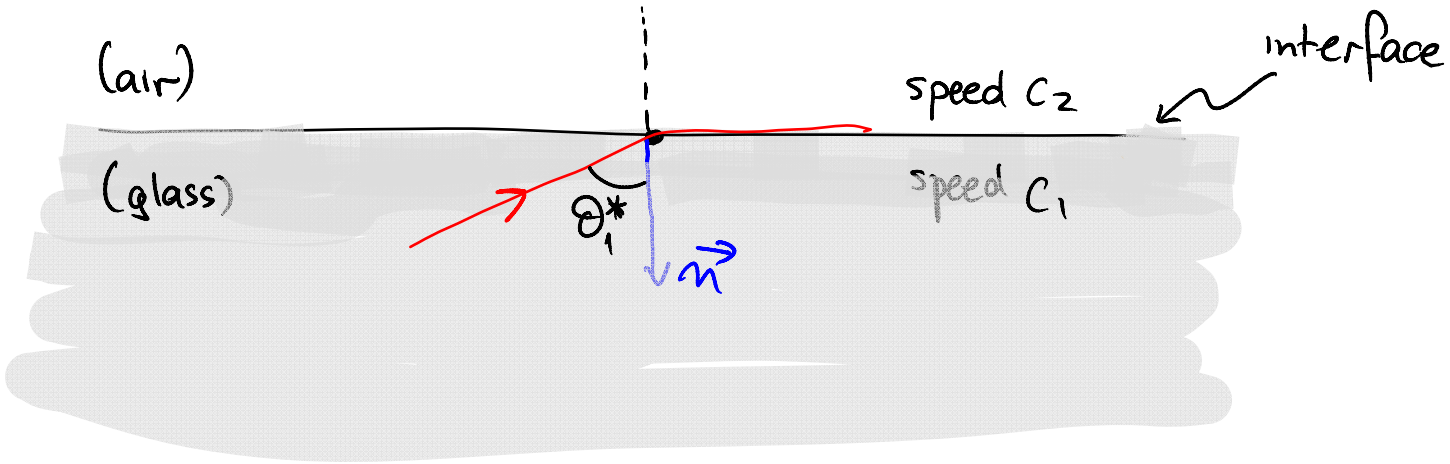
③ If $c_2 < c_1$ light bends toward the normal

If $c_2 > c_1$ light bends away from normal

Geometry of Refraction: The Critical Angle

Snell's law

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

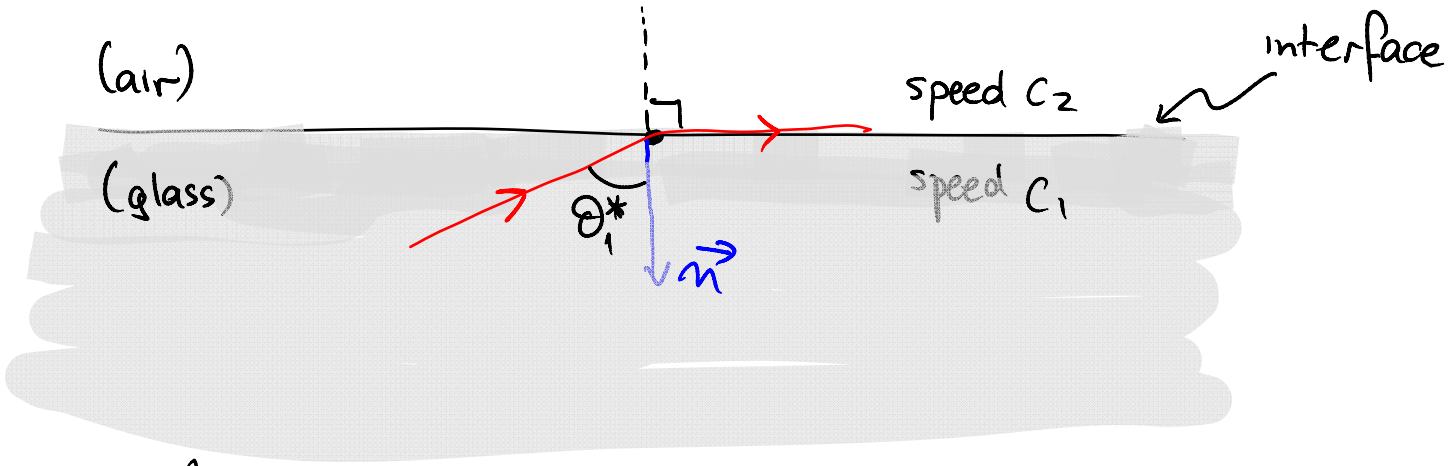


④ If $c_2 > c_1$ there is a critical angle above which no transmission occurs ($\Rightarrow$ have total internal reflection)

Geometry of Refraction: Total Internal Reflection

Snell's law

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



④ Deriving the critical angle: from Snell's law,

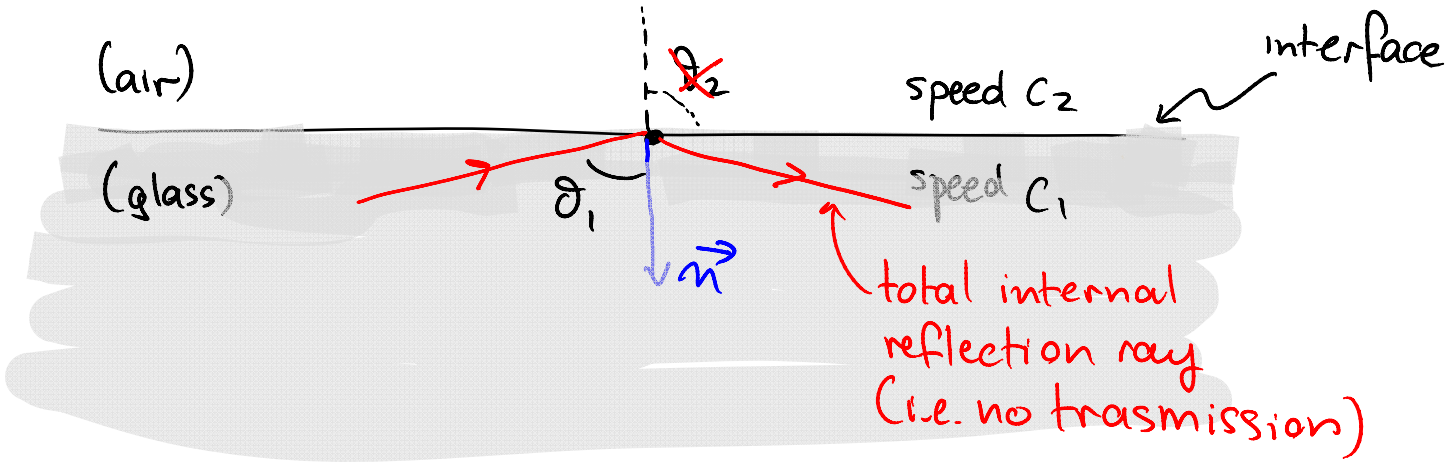
$$\cos\theta_2 = \sqrt{1-\left(\frac{c_2}{c_1}\right)^2 \sin^2\theta_1}$$

at critical angle, $\theta_2 = \frac{\pi}{2} \Rightarrow \sin\theta_1^* = \frac{c_1}{c_2}$

Geometry of Refraction: Total Internal Reflection

Snell's law

$$\frac{\sin\vartheta_1}{\sin\vartheta_2} = \frac{c_1}{c_2}$$

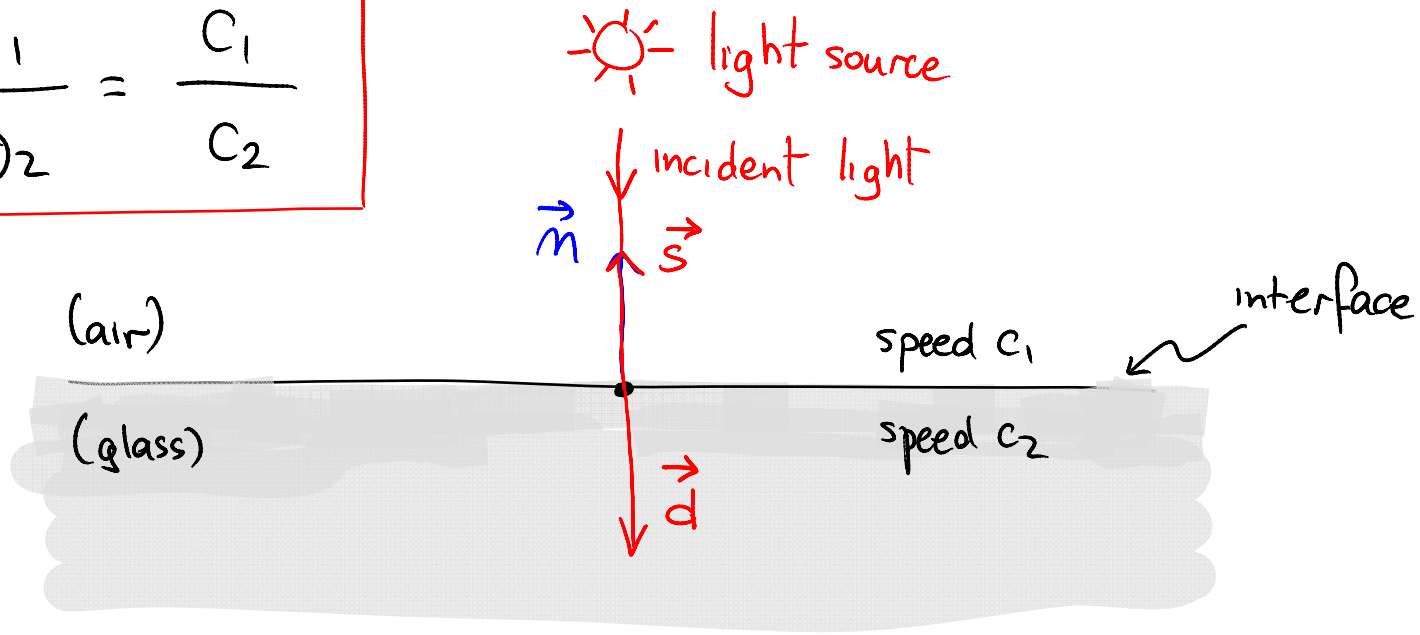


④ for $\vartheta_1 > \vartheta_1^*$, ϑ_2 is undefined

Geometry of Refraction: Normal Incidence

Snell's law

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



⑤ If $\theta_1 = 0$, no bending occurs

$$\vec{d} \text{ along } -\frac{c_2}{c_1}\vec{s} + \left[\frac{c_2}{c_1}\cos\theta_1 - \cos\theta_2\right]\vec{n}$$

Topic 12:

Basic Ray Tracing

- Introduction to ray tracing
- Computing rays
- Computing intersections
  - ray-triangle
  - ray-polygon
  - ray-quadric
  - the scene signature
- Computing normals
- Evaluating shading model
- Spawning rays
- Incorporating transmission
  - refraction
  - ray-spawning & refraction

Whitted Ray Tracing with Refraction

Basic idea:

- Spawn two rays
- One ray is along ideal specular direction
- One is along refraction direction

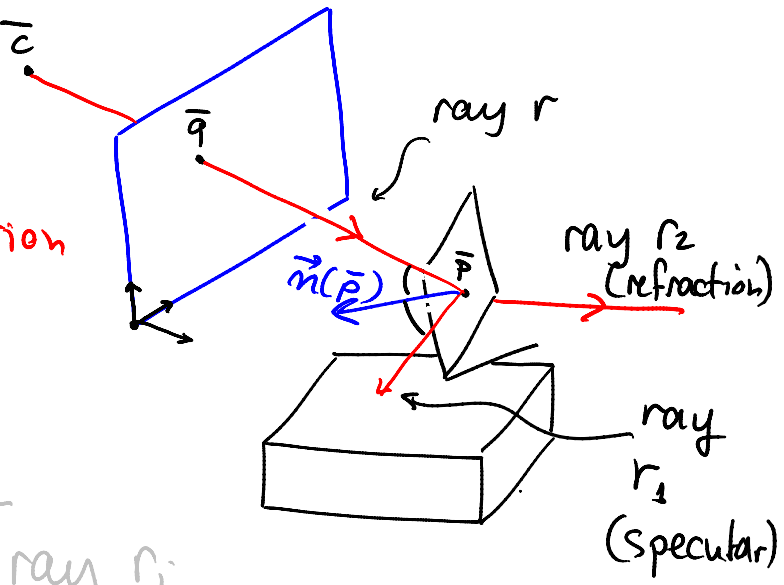
---

③ estimate amount of light reaching $\bar{p}$

a. "spawn" rays ~~$r_1, r_2, \ldots, r_k$~~ r_1, r_2 from $\bar{p}$ in ~~various~~ specular direction ~~directions~~ and refraction direction

b. if ray r_i hits a light source, estimate light travelling along r_i and stop

c. else apply loop recursively to ray r_i



Whitted Ray Tracing with Refraction

Much less efficient that specular-only ray tracing because 2^n rays are spawned after n bounces (instead of n)

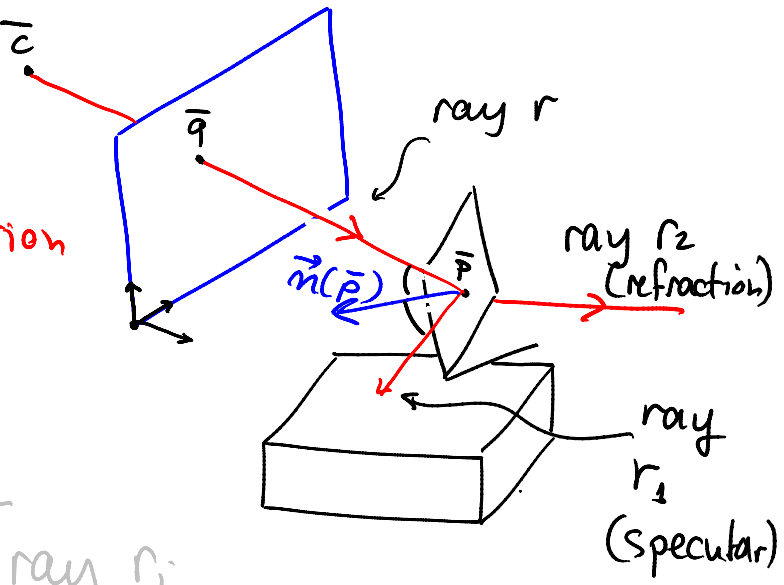
---

③ estimate amount of light reaching $\bar{p}$

a. "spawn" rays ~~$r_1, r_2, \ldots, r_k$~~ r_1, r_2 from $\bar{p}$ in ~~various~~ specular direction ~~directions~~ and refraction direction

b. if ray r_i hits a light source, estimate light travelling along r_i and stop

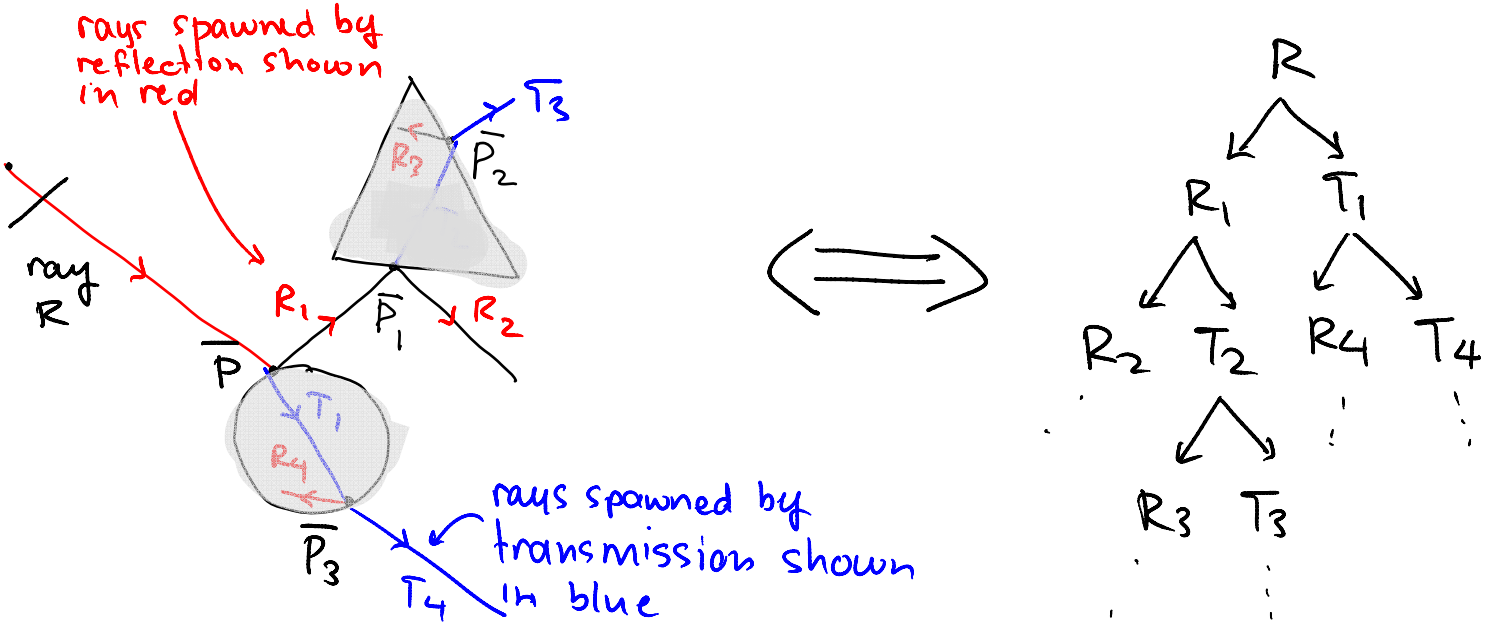
c. else apply loop recursively to ray r_i



Visualizing the Spawned Rays

Much less efficient that specular-only ray tracing because 2^n rays are spawned after n bounces (instead of n)

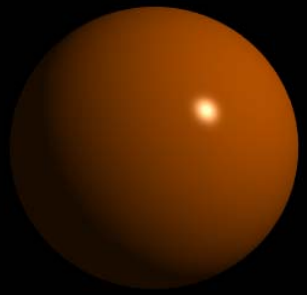
Visualizing ray-spawning as a tree:



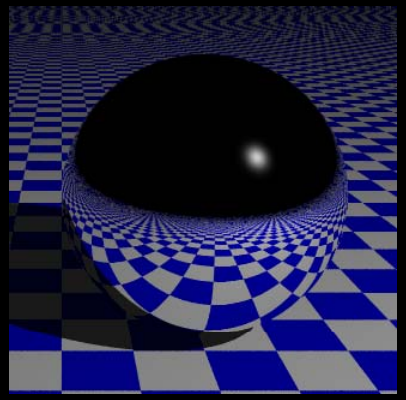
Local shading

Reflection

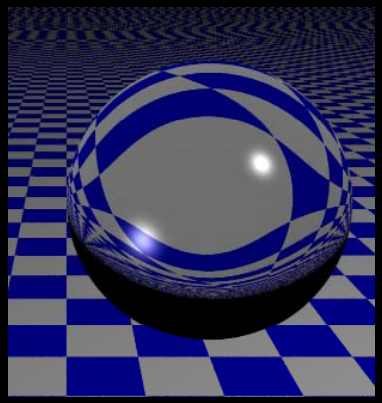
Transmission

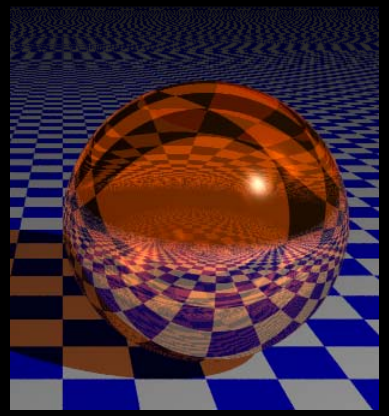


+



+





In-class lecture ends here

Lecture 8 ends here