Today's Topics

11. Texture mapping
12. Introduction to ray tracing

Topic 11:

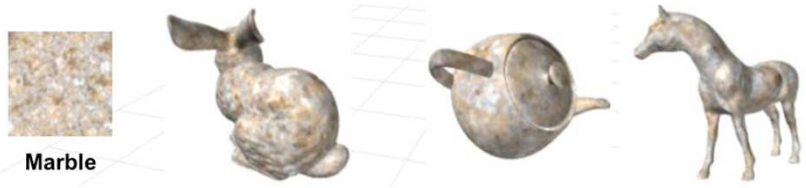
Texture Mapping

- 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


Gu et al, EGSR '07


Rendering w/ no subsurface scattering (opaque skin)
Jensen et al, SIGGRAPH '01

Texture Mapping: Motivation

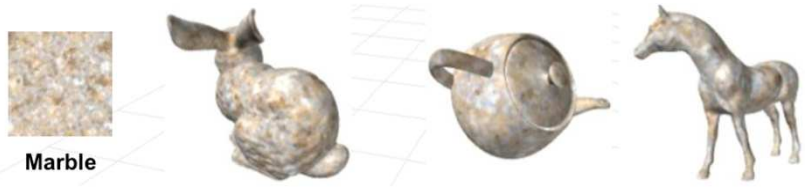


Goal: Endow objects with more varied & realistic appearance through complex variations in reflectance

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:

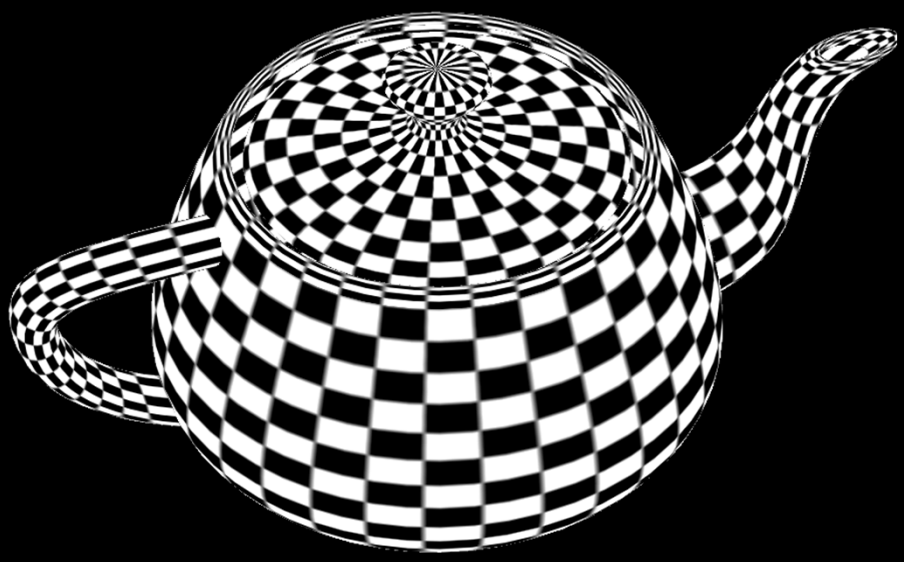
Texture Mapping

- 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

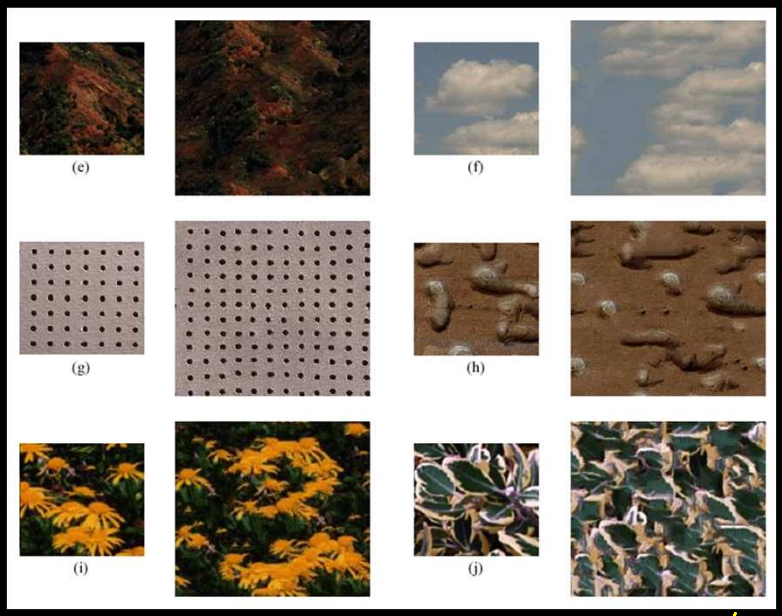
Photos of real materials

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

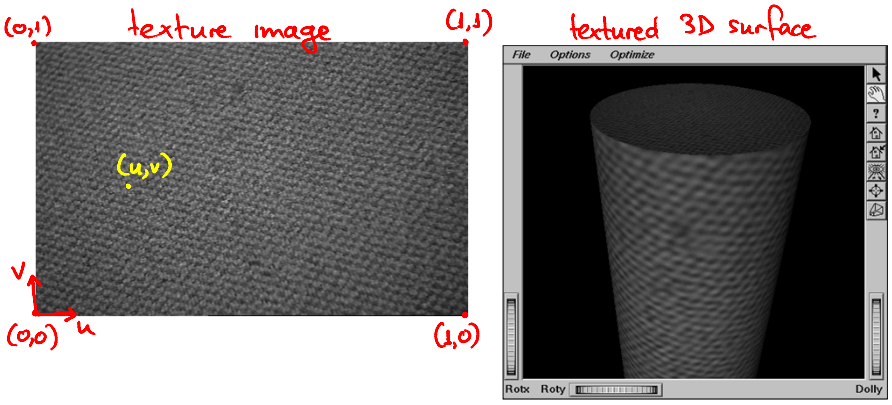
Kwatra et al, SIGGRAPH '05

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

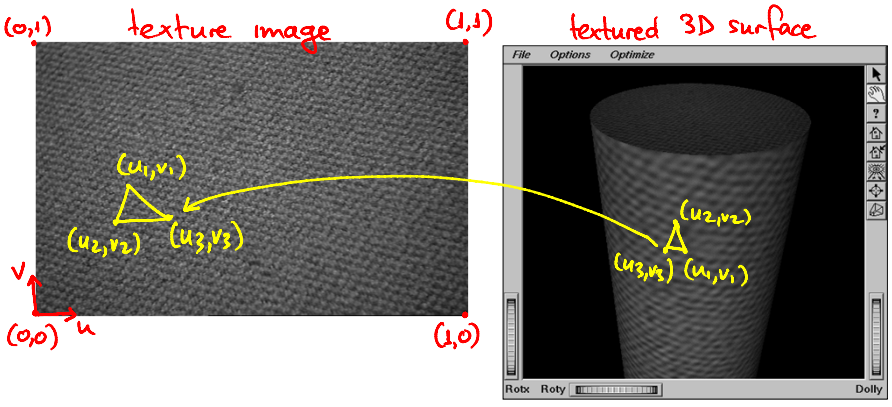
Texture Coordinates



Key ideas

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

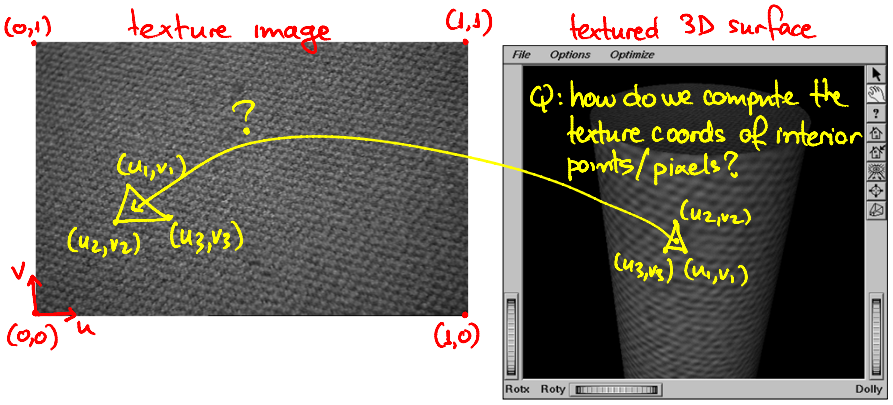
Specifying a Triangle's Texture Coordinates



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

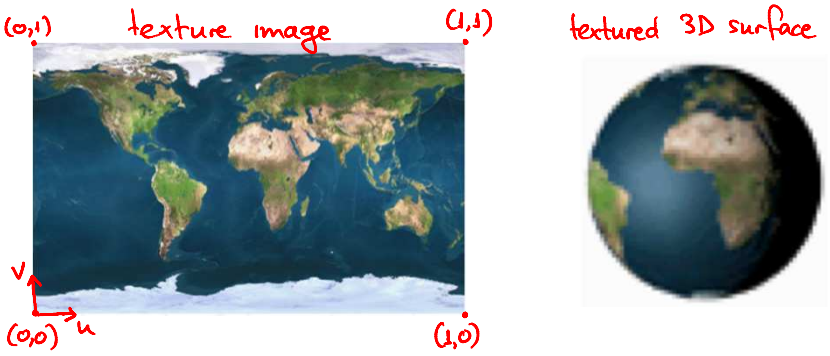
Texture Coords from Triangle Vertices



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

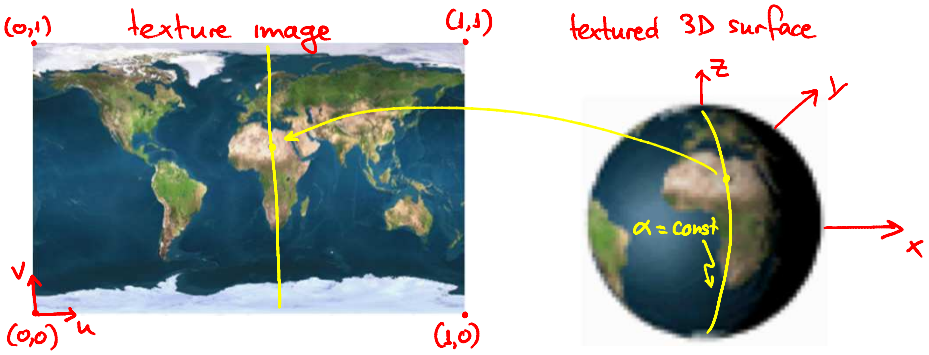
Texture Coords from Surface Parameters



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 $s(\alpha,\beta)$:

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

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

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

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

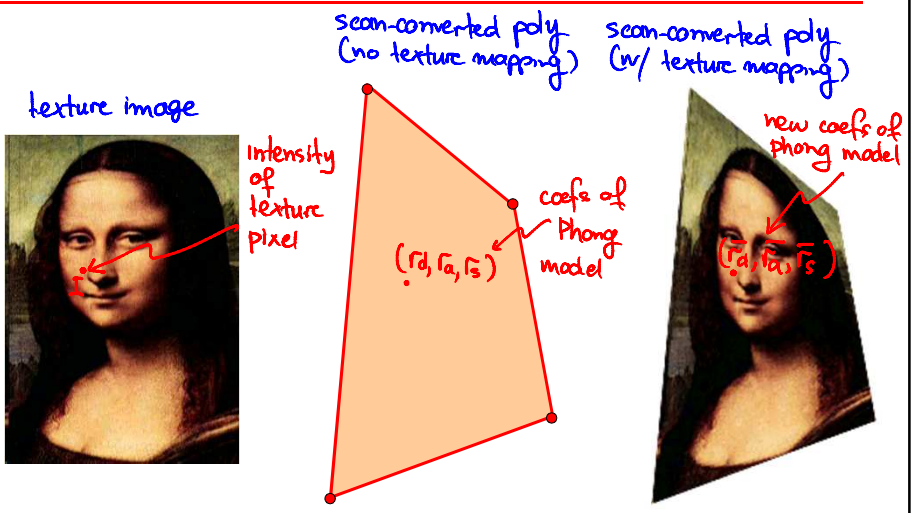
$$\beta \in [0, \pi)$$

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

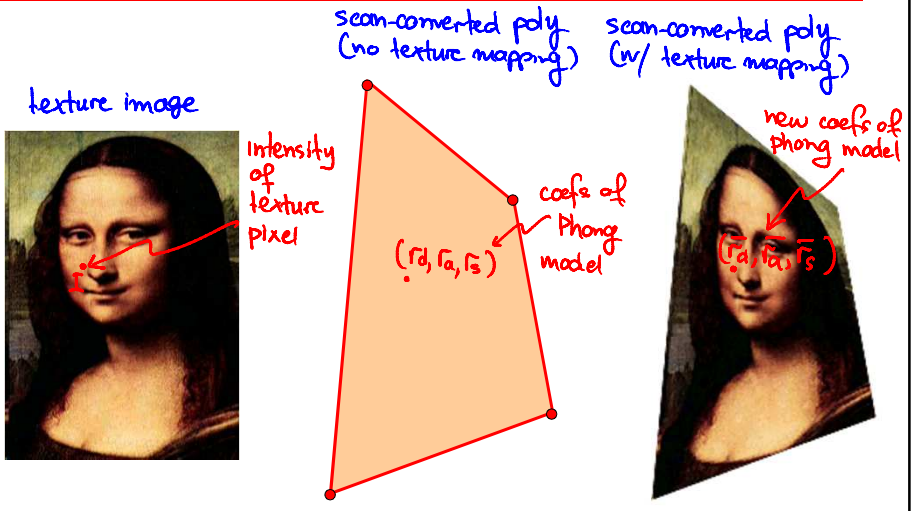
Appearance Control via Texture Mapping



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])

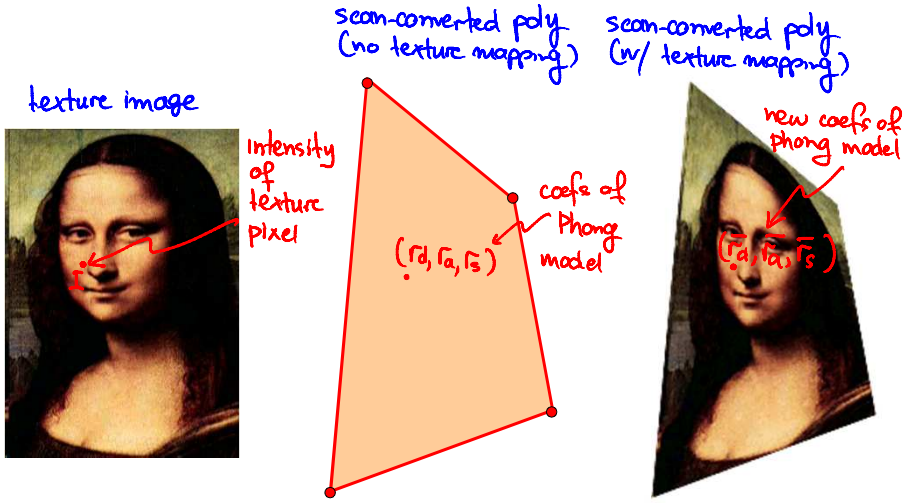
Appearance Control via Texture Mapping



Approach #2 (replacement)

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

Appearance Control via Texture Mapping



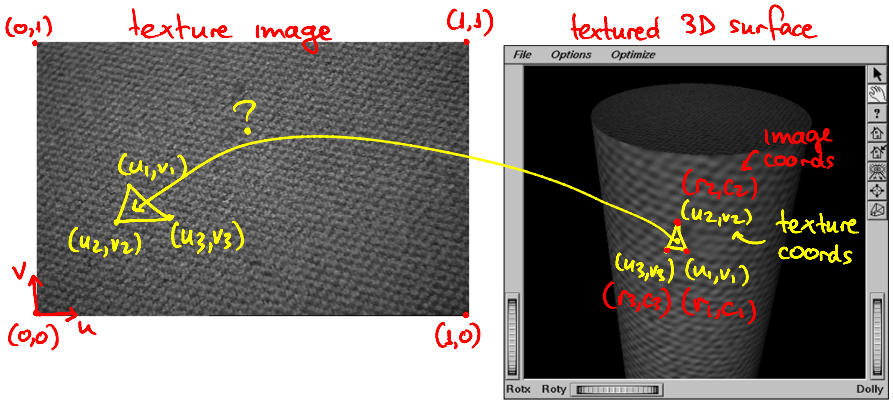
Other options also possible – see glTexEnvf() in OpenGL

Topic 11:

Texture Mapping

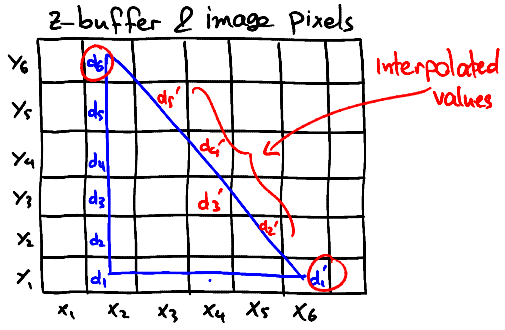
- Motivation
- Sources of texture
- Texture coordinates
- Controlling surface appearance with textures
- Texture mapping & scan conversion
- Perspectively-correct texture mapping
- Bump mapping, mip-mapping & env mapping

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



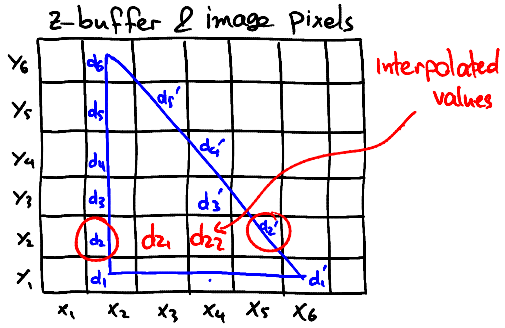
Step a:

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



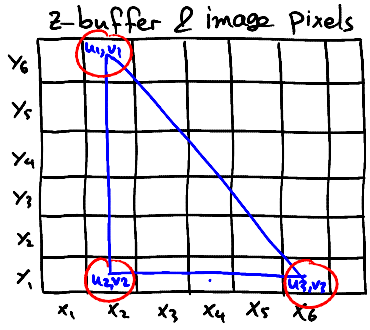
Step a:

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

Step b:

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

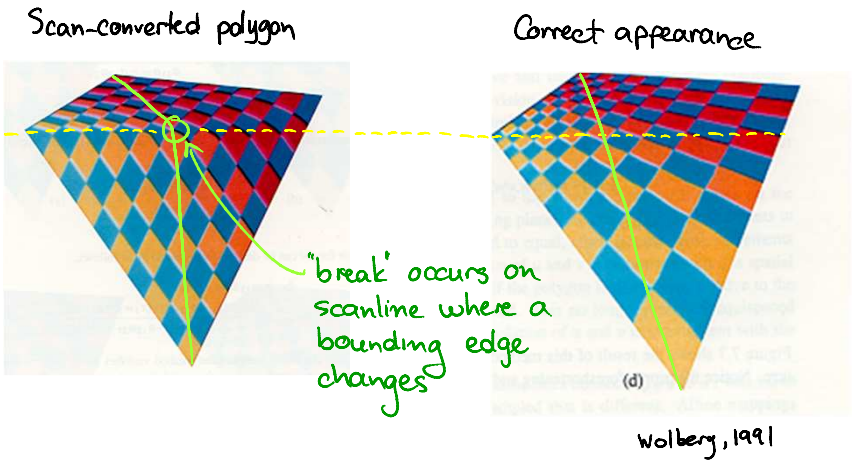
Should we Interpolate Texture Coordinates?



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

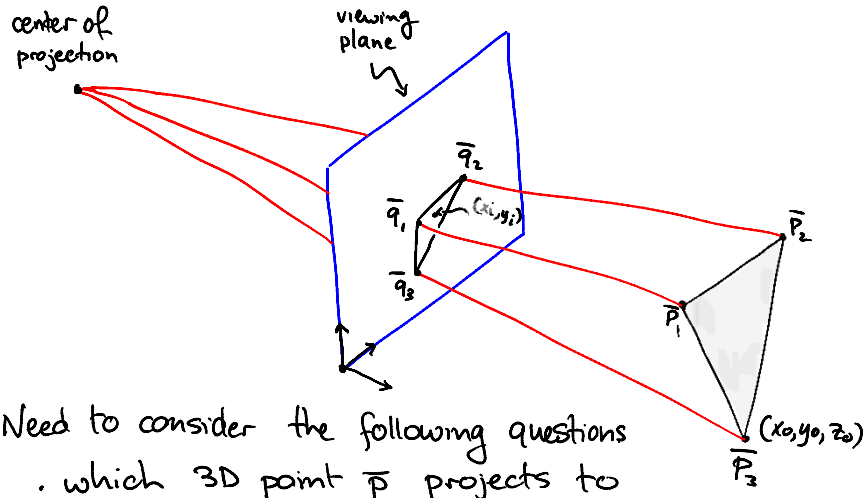
Not very well!

Artifacts Caused by Naive Scan Conversion



Wolberg, 1991

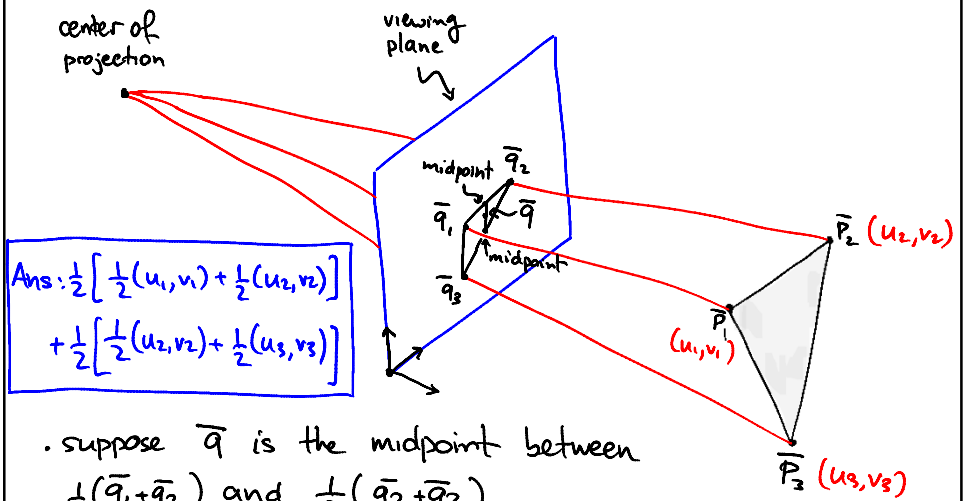
Why Do These Artifacts Occur?



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?



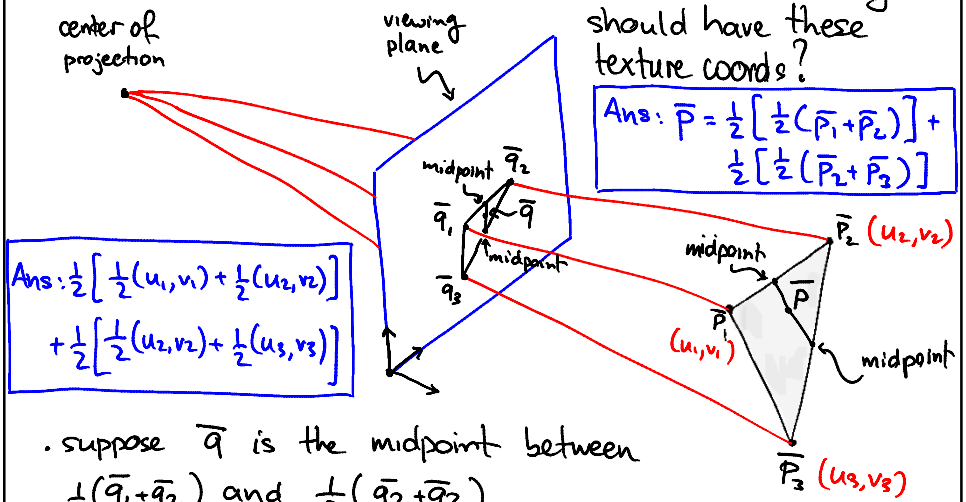
Ans: $\frac{1}{2}\left[\frac{1}{2}(u_1,v_1)+\frac{1}{2}(u_2,v_2)\right]$
$+\frac{1}{2}\left[\frac{1}{2}(u_2,v_2)+\frac{1}{2}(u_3,v_3)\right]$

- 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?

which pt on triangle should have these texture coords?

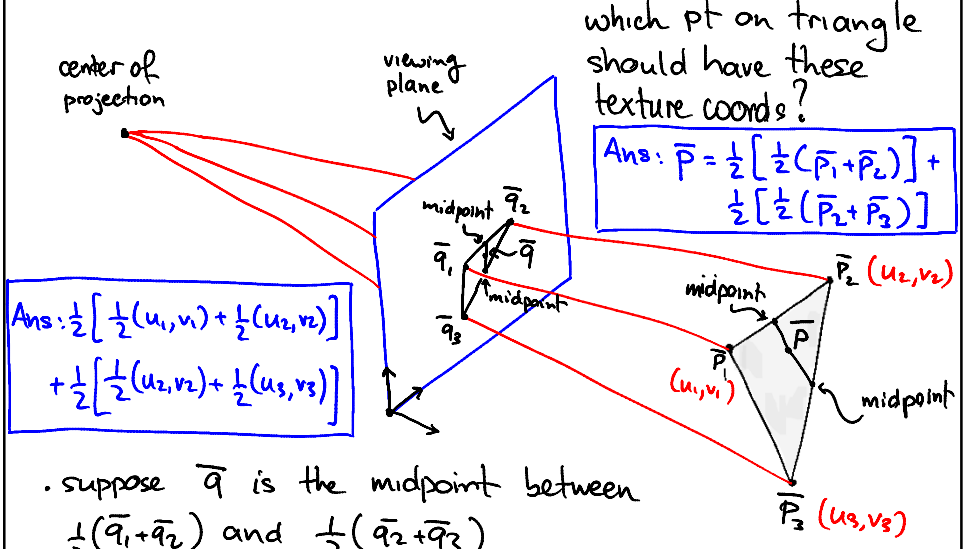
Ans: $\bar{P}=\frac{1}{2}\left[\frac{1}{2}(\bar{P}_1+\bar{P}_2)\right]+\frac{1}{2}\left[\frac{1}{2}(\bar{P}_2+\bar{P}_3)\right]$



Ans: $\frac{1}{2}\left[\frac{1}{2}(u_1,v_1)+\frac{1}{2}(u_2,v_2)\right]$
$+\frac{1}{2}\left[\frac{1}{2}(u_2,v_2)+\frac{1}{2}(u_3,v_3)\right]$

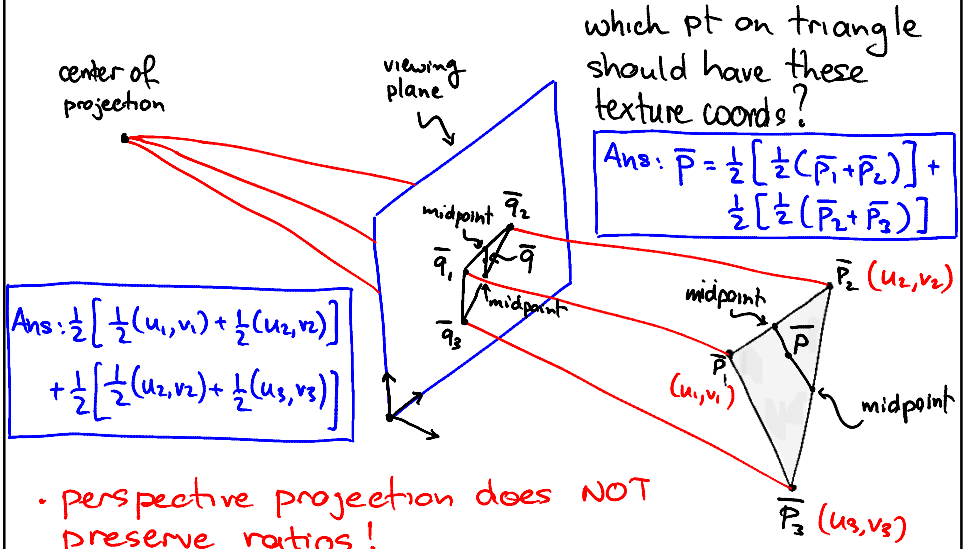
- suppose $\bar{q}$ is the midpoint between $\frac{1}{2}(\bar{q}_1+\bar{q}_2)$ and $\frac{1}{2}(\bar{q}_2+\bar{q}_3)$
- what texture coords will be assigned to $\bar{q}$?

Why Do These Artifacts Occur?



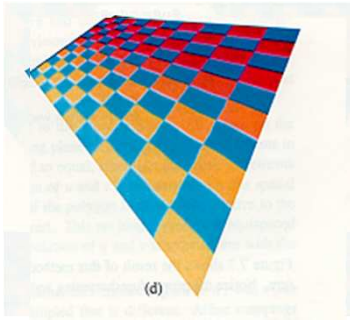
- suppose $\bar{q}$ is the midpoint between $\frac{1}{2}(\bar{q}_1+\bar{q}_2)$ and $\frac{1}{2}(\bar{q}_2+\bar{q}_3)$
- problem: $\bar{p}$ will <u>not</u> project to $\bar{q}$ in general

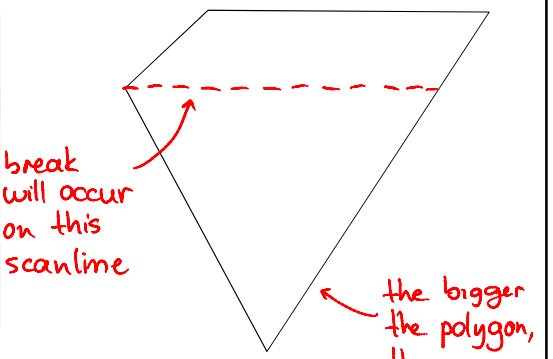
Why Do These Artifacts Occur?



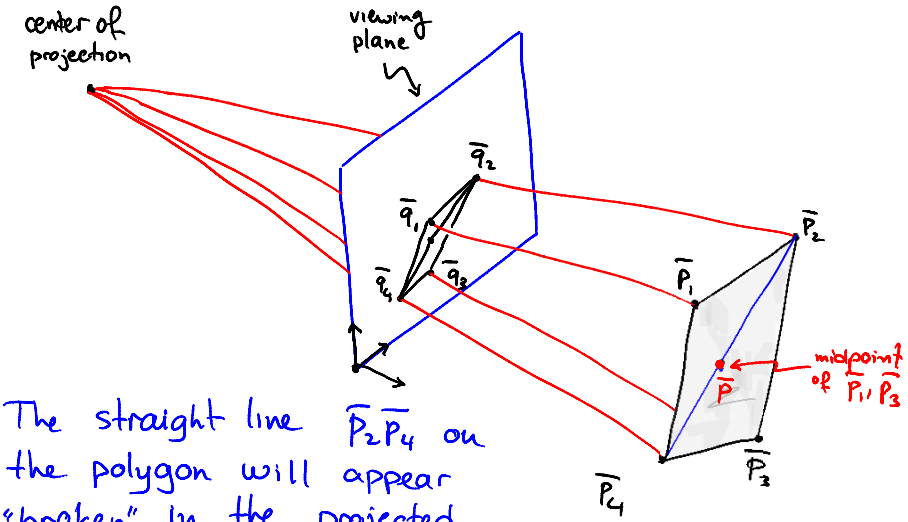
- perspective projection does NOT preserve ratios!
- problem: $\bar{p}$ will <u>not</u> project to $\bar{q}$ in general

Reducing Artifacts of Naive Scan Conversion



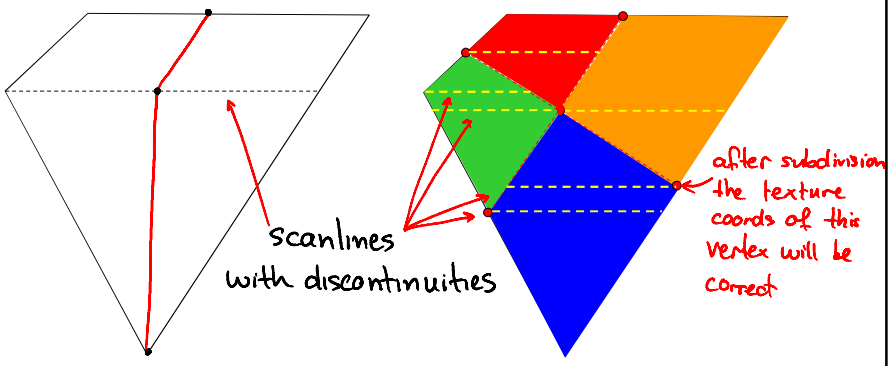


Reducing Artifacts of Naive Scan Conversion



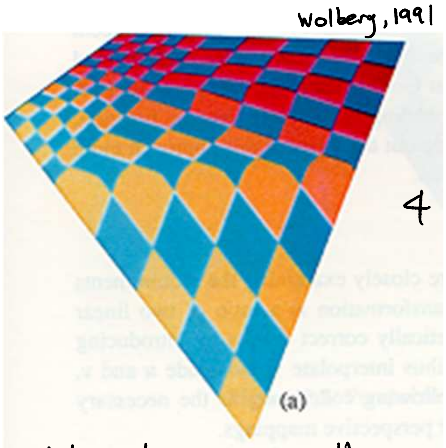
The straight line $\bar{P}_2\bar{P}_4$ on the polygon will appear "broken" in the projected triangle

Reducing Artifacts of Naive Scan Conversion



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

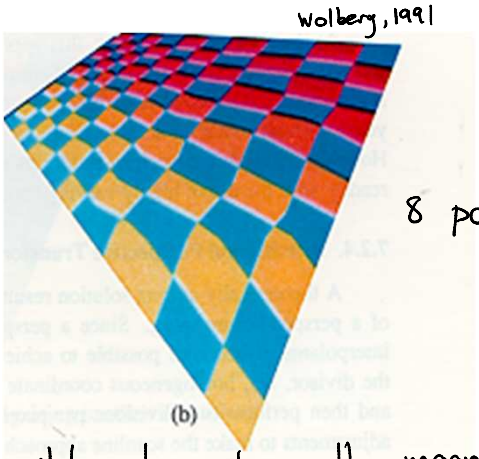
Reducing Artifacts by Polygon Subdivision



4 polys

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

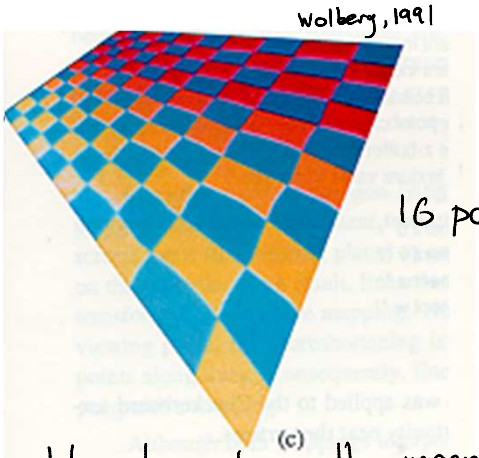
Reducing Artifacts by Polygon Subdivision



8 polys

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

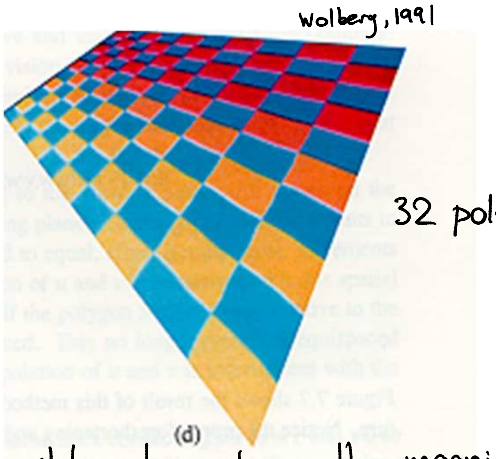
Reducing Artifacts by Polygon Subdivision



16 polys

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

Reducing Artifacts by Polygon Subdivision



32 polys

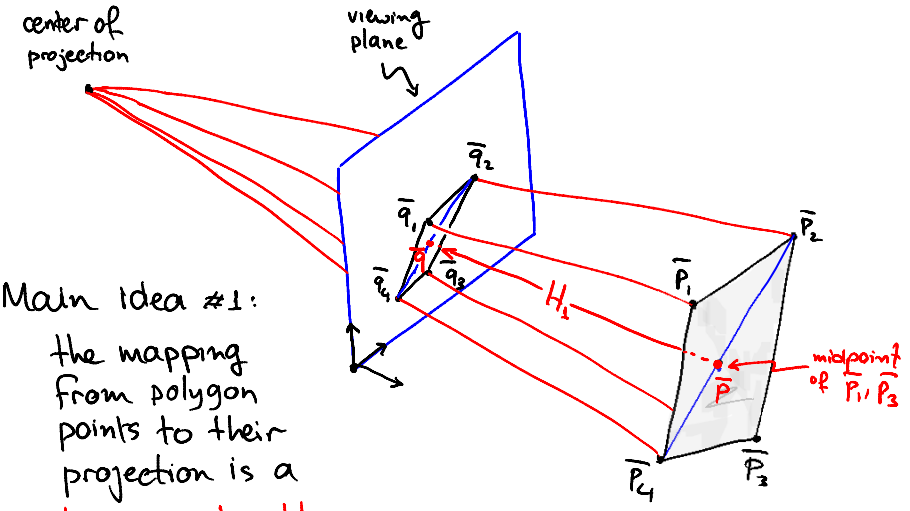
It is possible to reduce the magnitude of distortions by subdividing the polygon

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

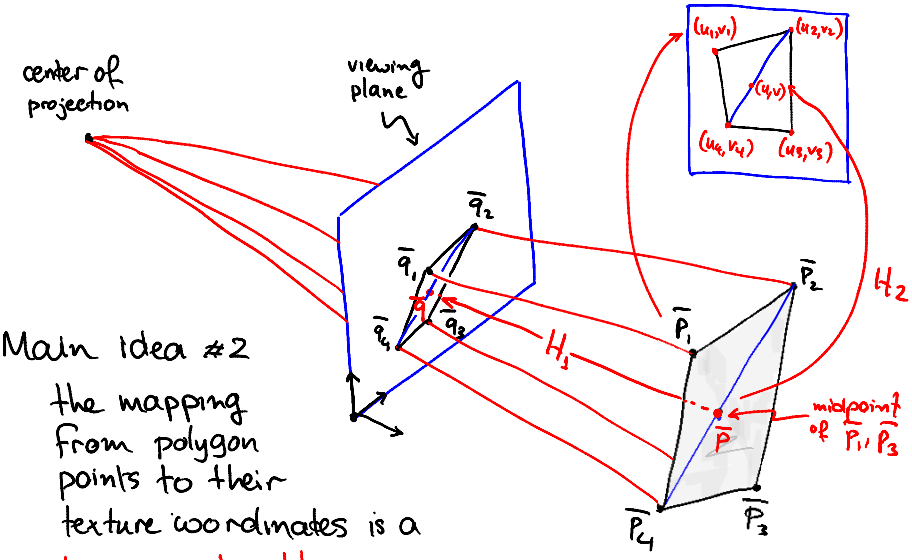
Perspectively-Correct Texture Mapping



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

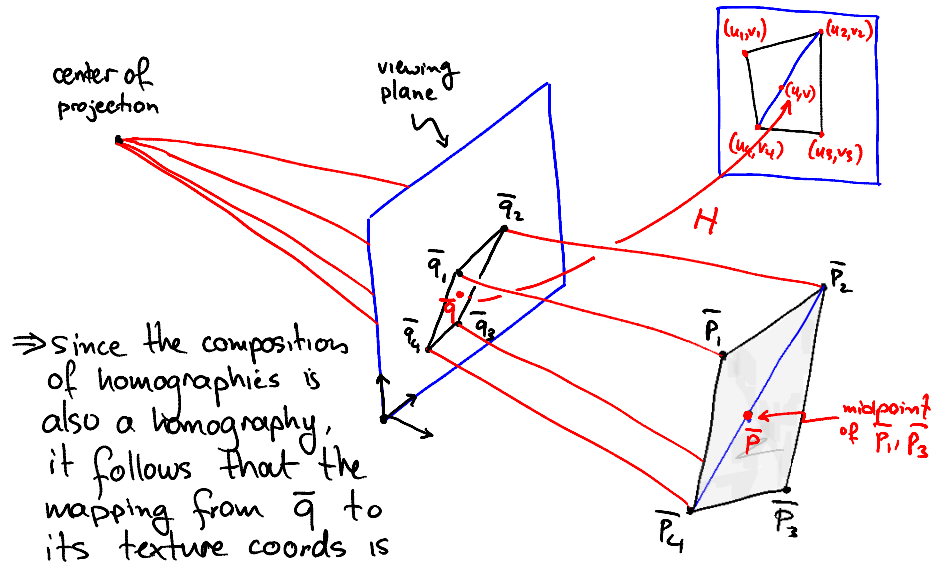
Perspectively-Correct Texture Mapping



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



⇒ since the compositions 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

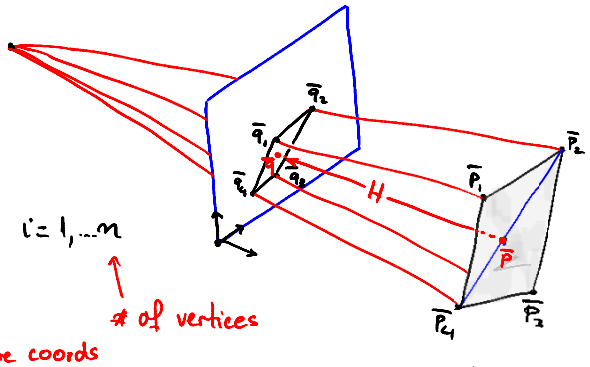
① Compute the homography H such that

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

$H\bar{q}_i$: projection of ith vertex in homogeneous 2D coords

$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$: texture coords of ith vertex express as homogeneous vectors

n: # of vertices



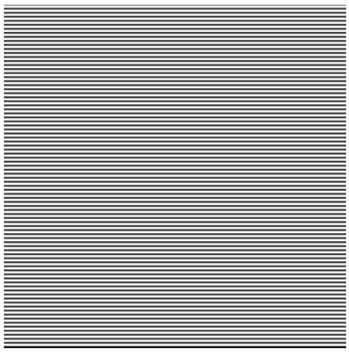
② Use H to compute the texture coords of a pixel $\bar{q} = (x_i, y_i, 1)$ in homogeneous coords

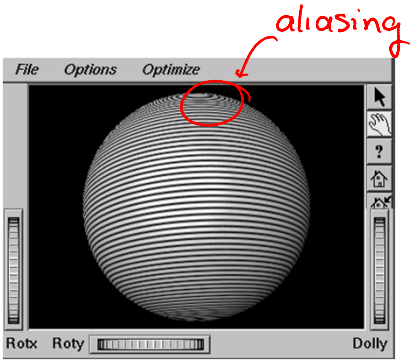
Topic 11:

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

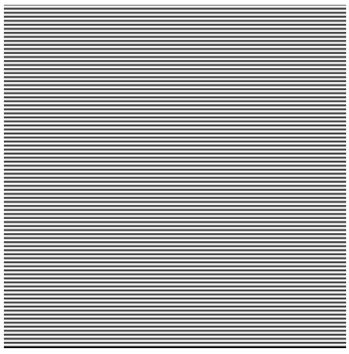
Aliasing During Texture Mapping

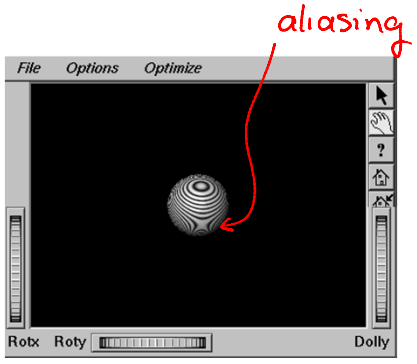




Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies")

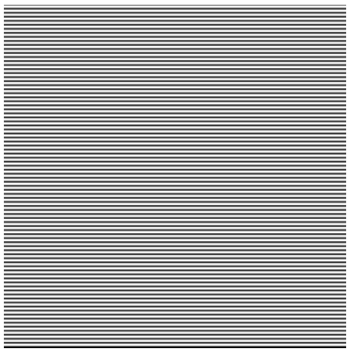
Aliasing During Texture Mapping

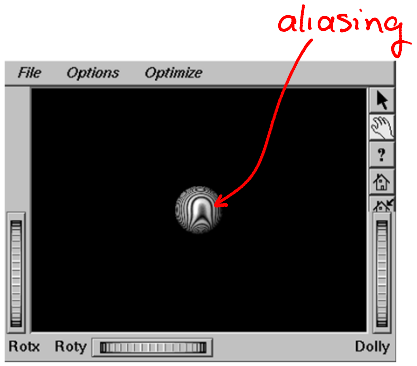




Texture mapping can produce significant aliasing artifacts when the texture images contain rapid variations (aka "high frequencies")

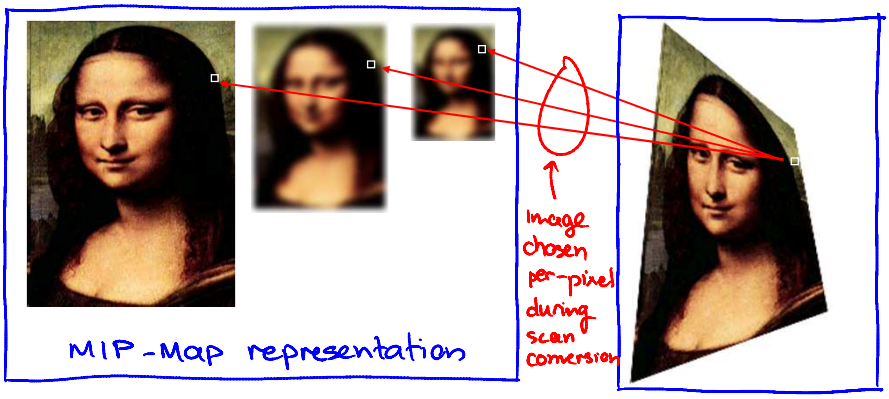
Aliasing During Texture Mapping





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

MIP-Mapping: Basic Idea



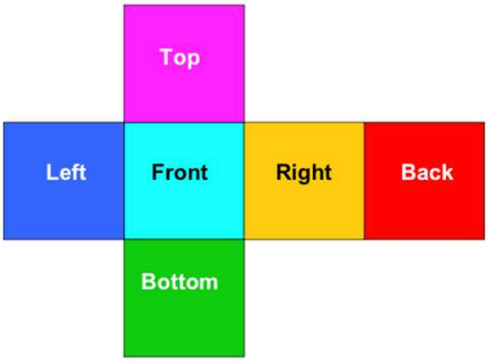
Solution: Use high-resolution texture for rendering objects that are close & low-res (i.e. blurred) texture when they are far away

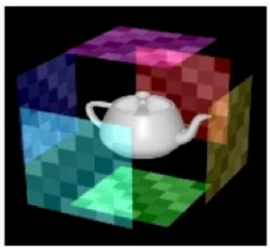
We can use textures that perturb normals instead of colors or reflectances

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

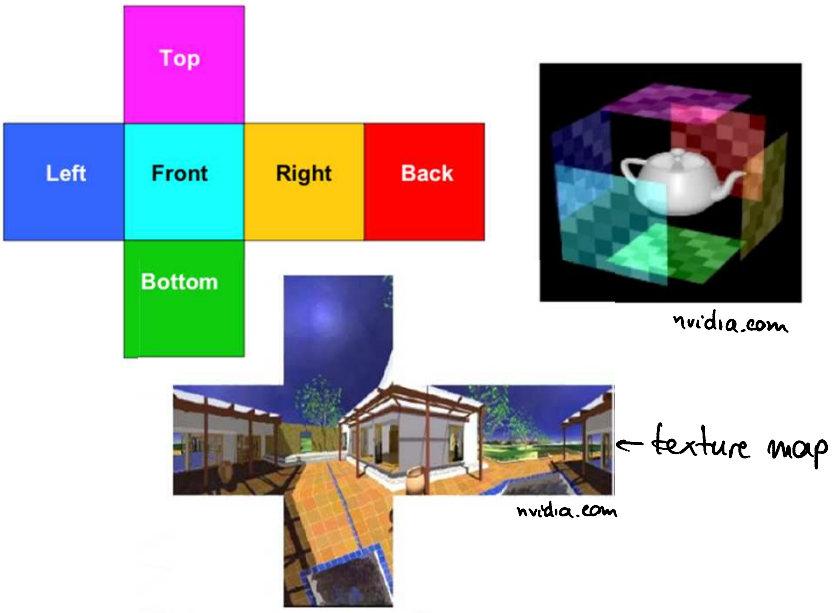
Environment Mapping



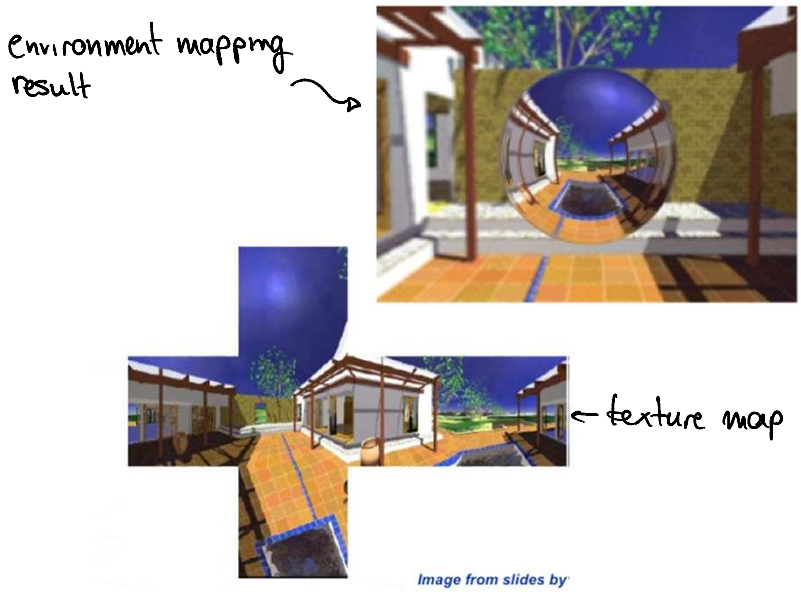


To create a full 360-degree environment map, we must texture all 6 interior faces of the cube

Environment Mapping



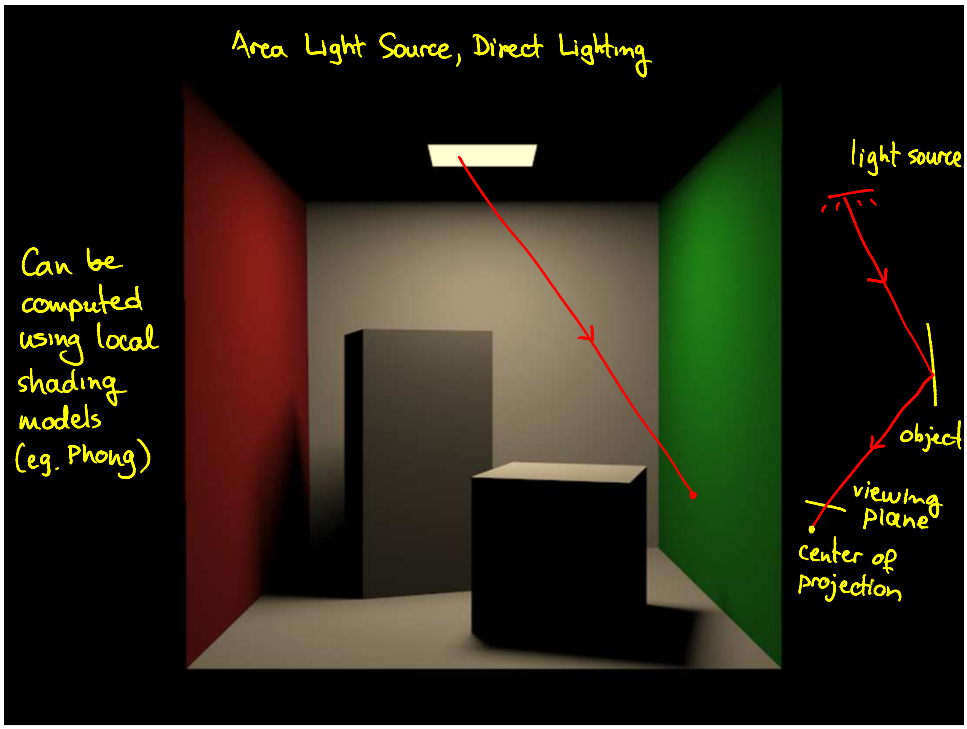
Environment Mapping

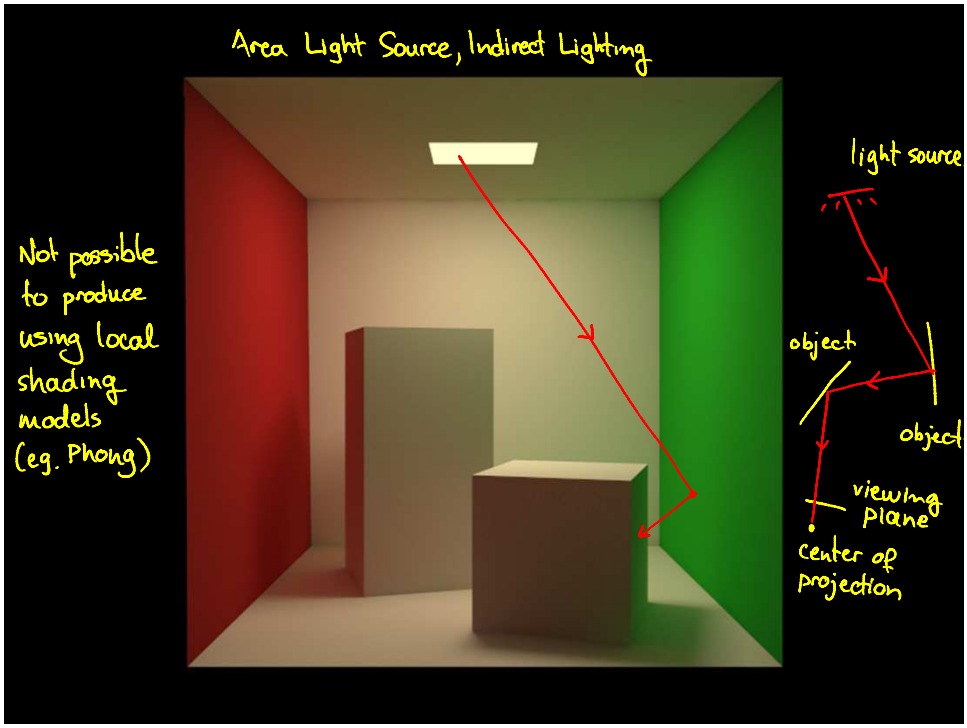


Topic 12:

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

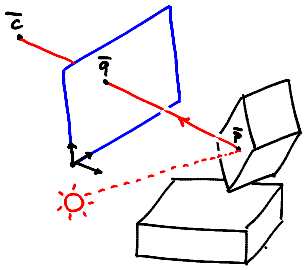




Rasterization vs. Ray Tracing

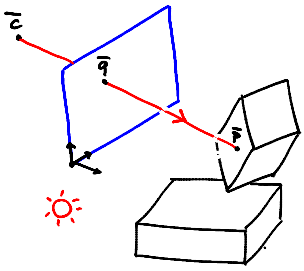
Rasterization:

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



Ray tracing:

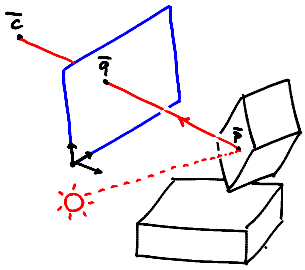
- Project pixels (aka "image samples") backwards onto scene
- Compute pixel color at $\bar{q}$ by estimating light reaching $\bar{p}$ directly or indirectly



Ray Tracing: Basic Idea

Rasterization:

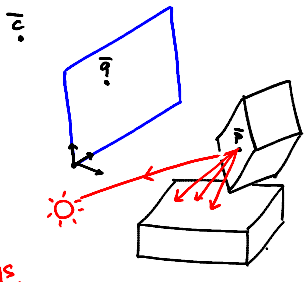
- Project geometry onto image
- Compute pixel color using local shading model



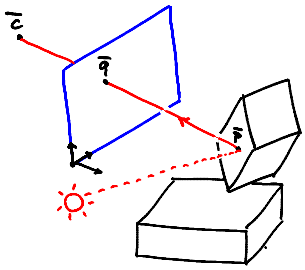
Ray tracing:

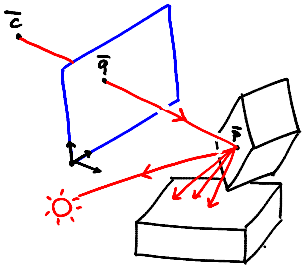
- Project pixels (aka "image samples") backwards onto scene
- Compute pixel color at $\bar{q}$ by estimating light reaching $\bar{p}$ directly or indirectly

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



Ray Tracing: Basic Idea

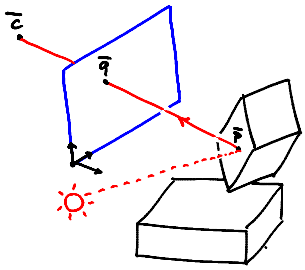


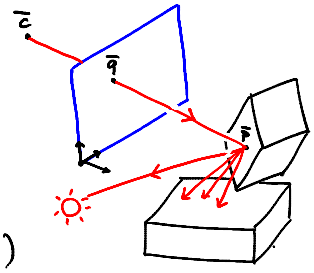


Ray Tracing: Advantages

- 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)

...



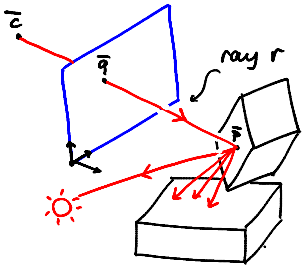


Ray Tracing: Basic Algorithm

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 $\bar{p}$)

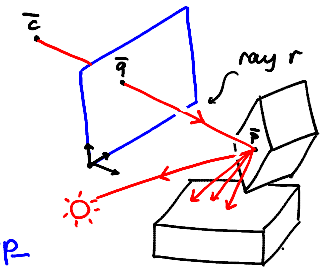
③ estimate amount of light reaching $\bar{p}$

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 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



Ray tracing in the movies

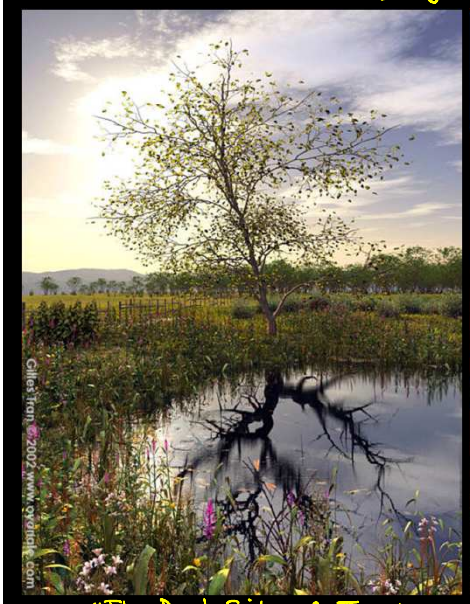
Christensen et al, 2006

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 ~ 8 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

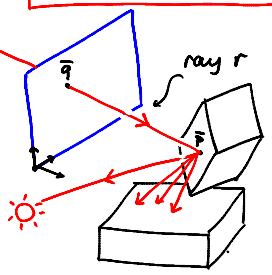
④ estimate amount of light travelling from $\bar{p}$ to $\bar{q}$ along ray r

defining the ray r

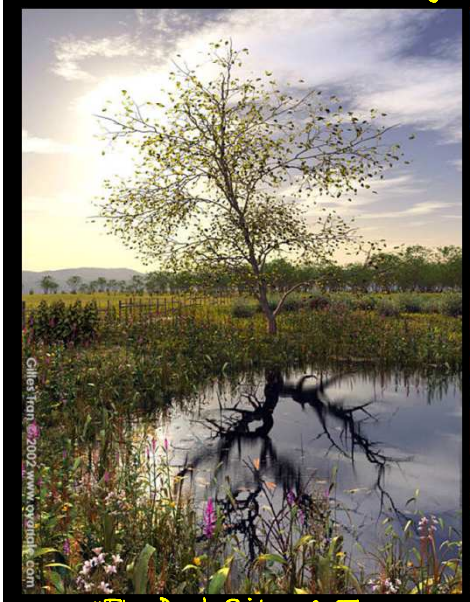
computing ray-scene intersections

evaluating reflectance model at $\bar{p}$

spawning rays



www.povray.org/community/hof



"The Dark Side of Trees"

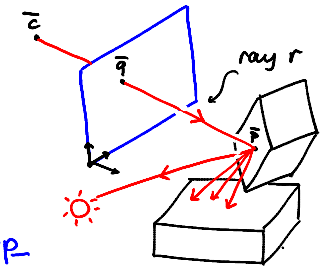
"Capriccio" B. Obukhov et al, 2003

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 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

Computing the Ray Through a Pixel

Basic loop:

for each pixel $\bar{q}$ — defining the ray r

① cast ray r through $\bar{q}$

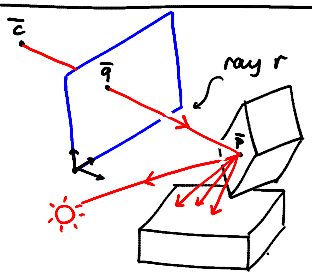
② find 1st intersection of $\bar{q}$ with scene (i.e. point $\bar{p}$)

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

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

Compute ray r given

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



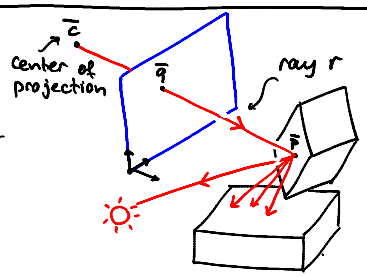
Computing the Ray Through a Pixel: Main Idea

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 $\bar{q}_w$?

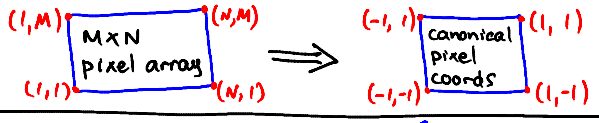
Compute ray r given
- discrete pixel position
- world-to-camera matrix M_{wc}
- camera-to-canonical new matrix M_{cv}



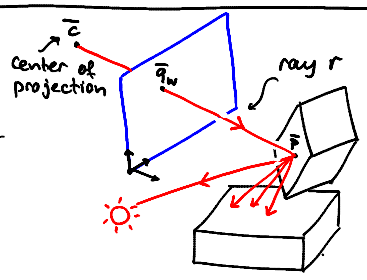
Computing the Ray Through a Pixel: Steps

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 new matrix M_{cv}



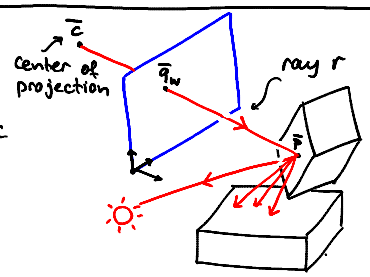
Computing the Ray Through a Pixel: Steps

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}