cs155 – z sweedyk

ray tracing

ray tracing

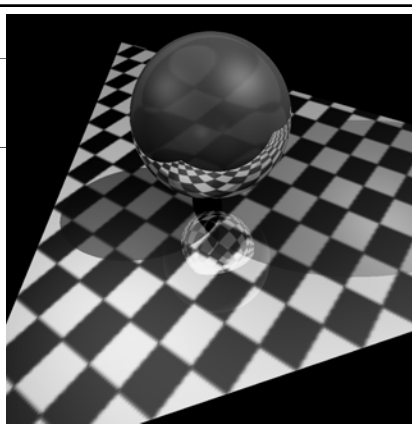
- simple ray casting
- recursive ray tracing
- **cheap tricks**
- optimizations

cheap tricks

- texture mapping

texture mapping

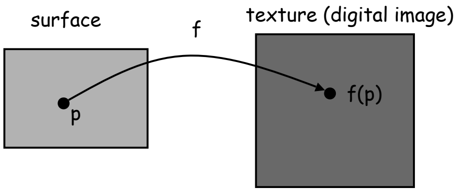
"glue" image to surface



steve yan, fall 2001

drew levin, fall 2001

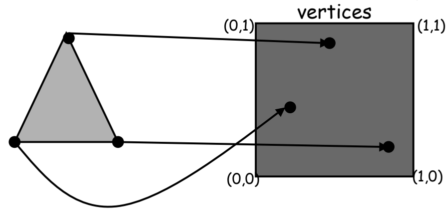
texture mapping



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

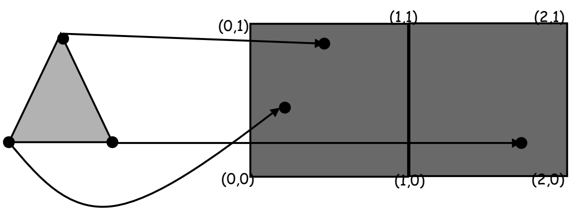
texture mapping triangle

Input specifies texture coordinates of triangle vertices

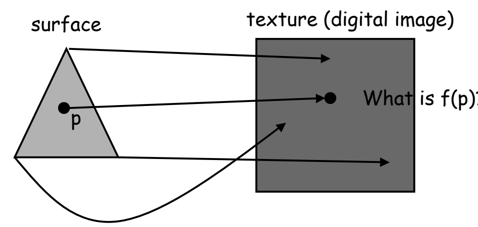


texture mapping triangle

texture coordinates need not be in [0,1]!

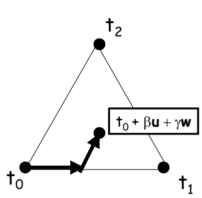


texture mapping triangle

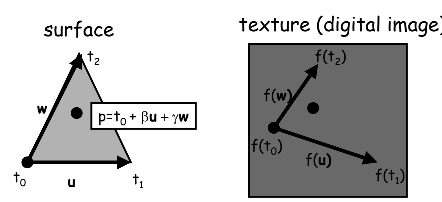


triangle: parametric form

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

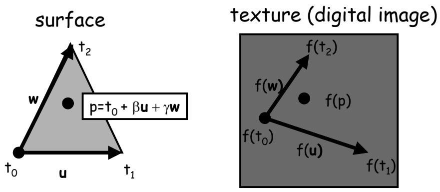


computing f(p)



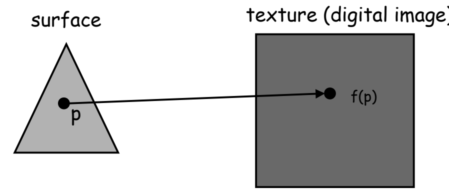
compute $f(\mathbf{u}) = f(t_1)-f(t_0)$ and $f(\mathbf{w})=f(t_2)-f(t_0)$

computing f(p)



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

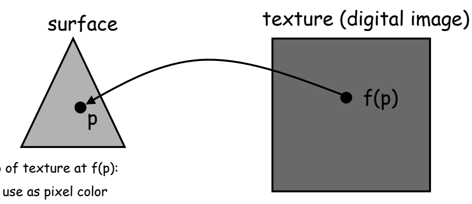
texture mapping triangle



What is image color at f(p)?

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

using the color



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

1. use as pixel color
2. use as diffuse and specular coefficient of surface at p
3. use as diffuse coefficient of surface at p
4. use as scaling factor to non-specular color due to direct illumination

DO THIS!

color

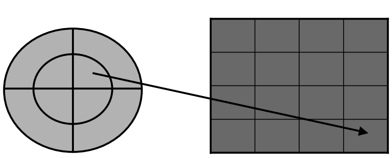
for each channel we'll approximate the color at the intersection point as the sum of five terms

- emission
- ambient reflection
- diffuse reflection
- specular reflection
- specular transmission

$\left.\begin{array}{l}\text{emission}\\ \text{ambient reflection}\\ \text{diffuse reflection}\end{array}\right\}$ scale these by texture color

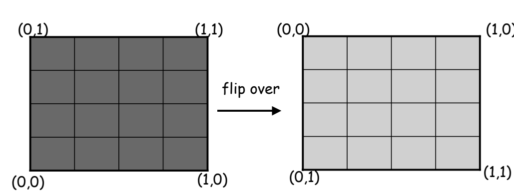
texture mapping sphere

e.g. latitude/longitude



conceptually: wrap texture

top down view

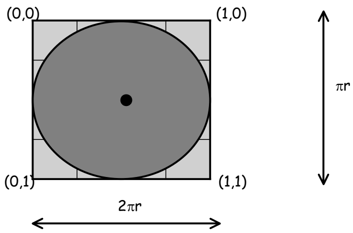


conceptually: wrap texture

top down view

we need wider paper

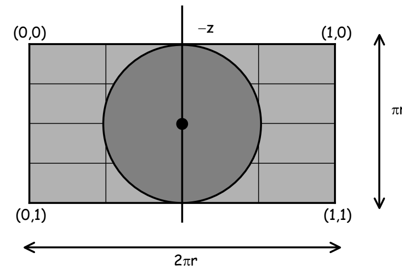
+y is coming towards us



conceptually: wrap texture

top down view

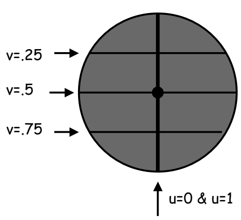
+y is coming towards us



conceptually: wrap texture

top down view

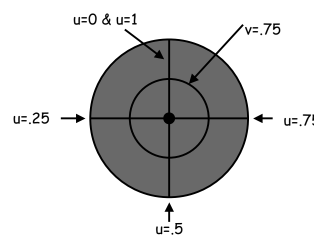
+y is coming towards us



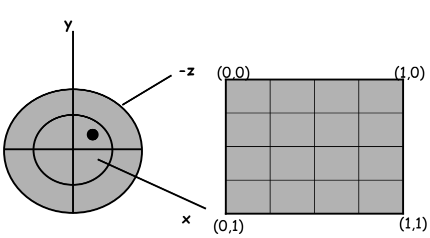
conceptually: wrap texture

front view

+z is coming towards us



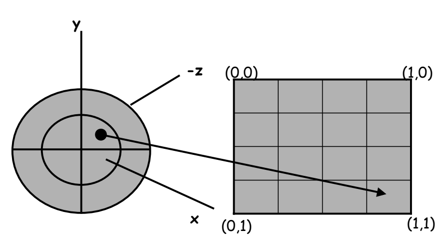
texture mapping sphere



intersection point P

has texture coordinate (u,v)

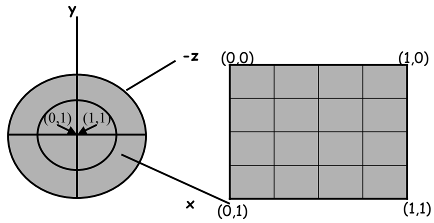
texture mapping sphere



intersection point P

has texture coordinate (u,v)

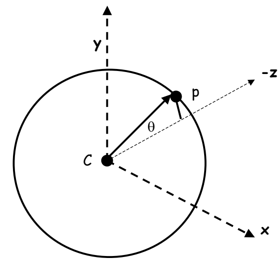
texture mapping sphere



intersection point P

has texture coordinate (u,v)

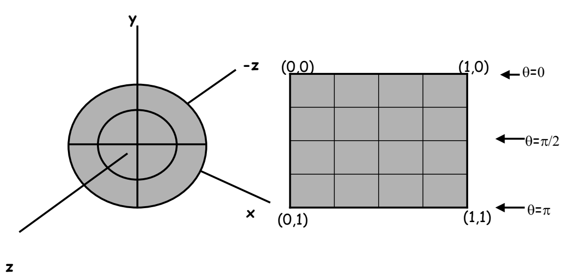
parameterization



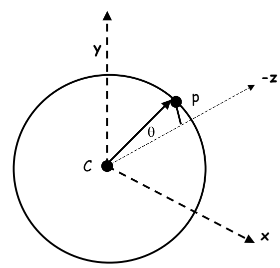
Assume P = C + a**x** + b**y** + g**z**

Let the angle between P-C and -**z** be θ

texture mapping sphere



parameterization

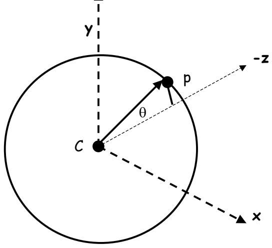


Assume P = C + a**x** + b**y** + g**z**

Let the angle between P-C and **z** be θ

What is relationship between g and θ ?

parameterization



Assume P = C + a**x** + b**y** + g**z**

Let the angle between P-C and **z** be θ

$-g = r \cos\theta$. So...

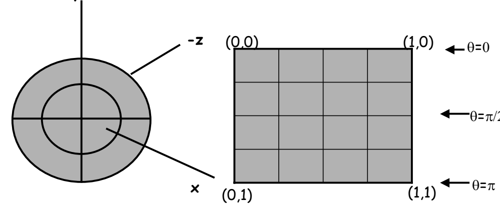
If $g = 0$ then $\theta = \pi/2$.

If $g \neq 0$ then $\theta = \arccos(-g/r)$.

Implementation note: arccos returns $\theta \in [0,\pi]$.

texture mapping sphere

Implementation note: arccos returns $\theta \in [0,\pi]$.

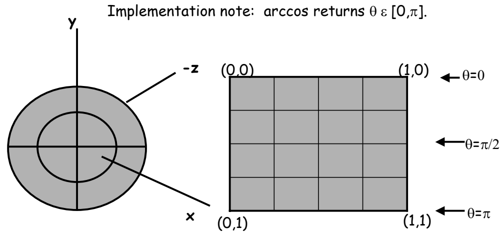


intersection point P = C + a**x** + b**y** + g**z**

has texture coordinate (u, v) where v=_______

texture mapping sphere

Implementation note: arccos returns θ ε [0,π].

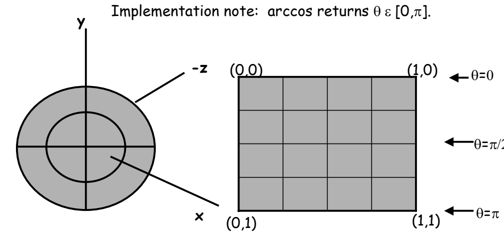


intersection point P = *C* + a**x** + b**y** + g**z**

has texture coordinate (u, v) where v=θ/π

texture mapping sphere

Implementation note: arccos returns θ ε [0,π].



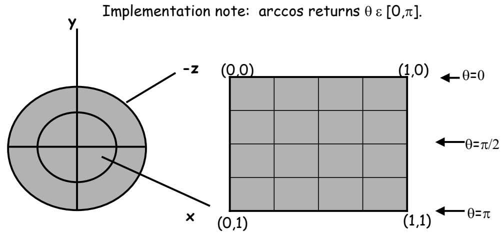
intersection point P = *C* + a**x** + b**y** + g**z**

has texture coordinate (u, v) where v=θ/π

if θ=0 or θ=π then u=_________

texture mapping sphere

Implementation note: arccos returns θ ε [0,π].

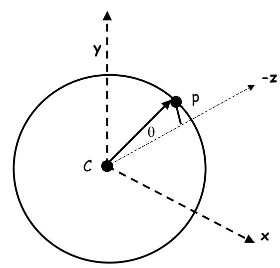


intersection point P = *C* + a**x** + b**y** + g**z**

has texture coordinate (u, v) where v=θ/π

if θ=0 or θ=π then u=1/2 else_____________

parameterization

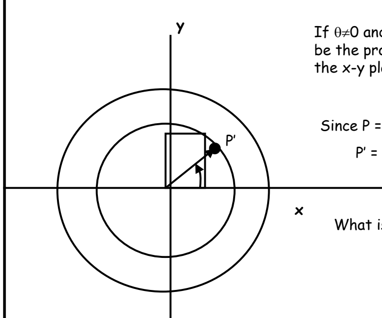


If θ≠0 and θ≠π then let P'-*C* be the projection of P-*C* on the x-y plane.

Since P = *C* + a**x** + b**y** + g**z**

P'=____________

parameterization



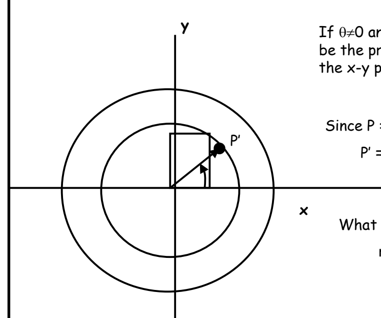
If θ≠0 and θ≠π then let P'-*C* be the projection of P-*C* on the x-y plane.

Since P = *C* + a**x** + b**y** + g**z**

P' = *C* + a**x** + b**y**

What is length of P'?

parameterization



If θ≠0 and θ≠π then let P'-*C* be the projection of P-*C* on the x-y plane.

Since P = *C* + a**x** + b**y** + g**z**

P' = *C* + a**x** + b**y**

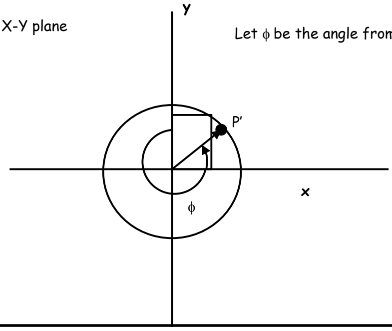
What is length of P'?

r'= r sin θ

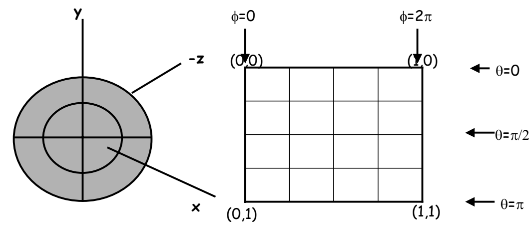
parameterization in x-z plane

X-Y plane

Let ϕ be the angle from **y** to P'-C CCW



texture mapping sphere



intersection point P = C + a**x** + b**y** + g**z**

has texture coordinate (u, v) where v=θ/π

if θ=0 or $\theta=\pi$ then u=1/2

otherwise u=$\phi/(2\pi)$

parameterization in x-y plane

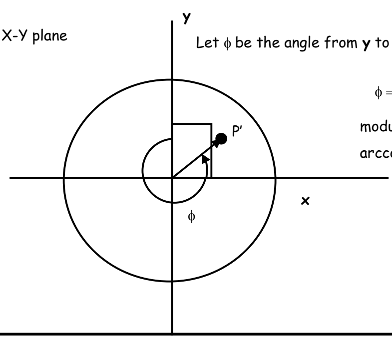
X-Y plane

Let ϕ be the angle from **y** to P'-C CCW from above

ϕ = arccos (b/r')

modulo the fact that

arccos($\bullet$) ε [0,π]



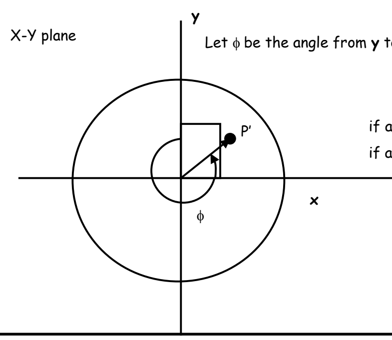
parameterization in x-y plane

X-Y plane

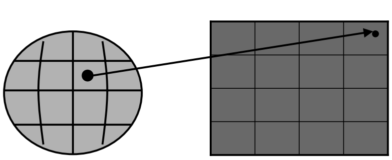
Let ϕ be the angle from **y** to P'-C CCW from above

if a≤0 ϕ = arccos (b/r')

if a>0 ϕ = 2π -arccos (b/r')



texture mapping sphere



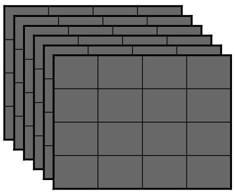
of course (u,v) won't typically be at sample points so use bilinear re-sampling

use texture color to scale ambient, emissive, and diffuse reflections

seams

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

3d textures



use stack of images

how do we generate these images?

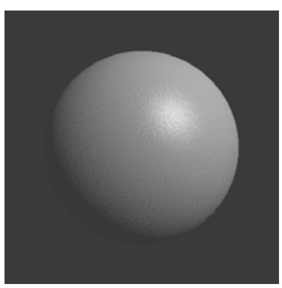
procedural textures: we'll come back to this

cheap tricks

- texture mapping
- bump mapping

creating bumpy surfaces

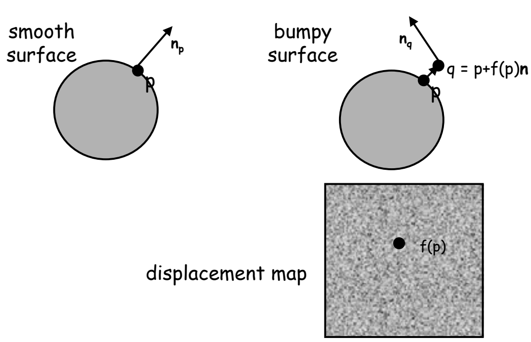
adrian mettler,
spring 2003



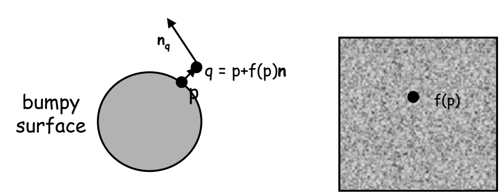
bump mapping vs texture mapping

- bump mapping is computationally more difficult that texture mapping
- bump mapping creates a better 3d look than texture mapping

smooth vs. bumpy surface models

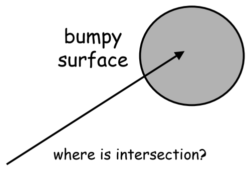


displacement mapping



displacement map=height field

displacement mapping problem

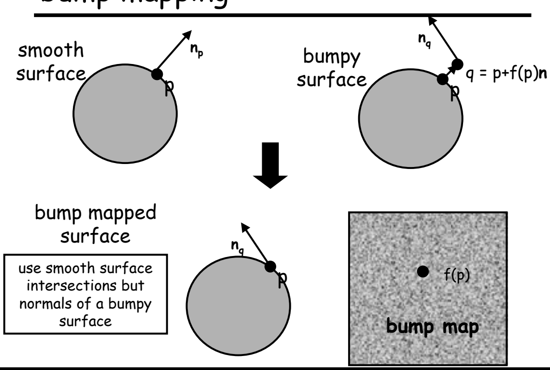


bumpy surfaces

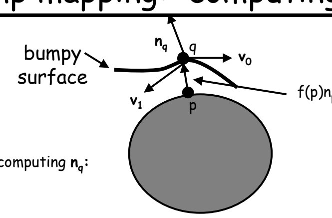
a surface appears to be bumpy because
- jagged silhouette
- surface normals fluctuate dramatically

Bump mapping: simulate this by perturbing normals in the lighting calculations

bump mapping



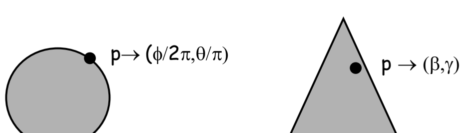
bump mapping: computing n_q



computing $\mathbf{n}_q$:

1. find vectors $\mathbf{v}_0$ and $\mathbf{v}_1$ in plane tangent to bumpy surface at point q (i.e. take derivatives)
2. $\mathbf{n}_q = (\mathbf{v}_0 \times \mathbf{v}_1)/||\mathbf{v}_0 \times \mathbf{v}_1||$

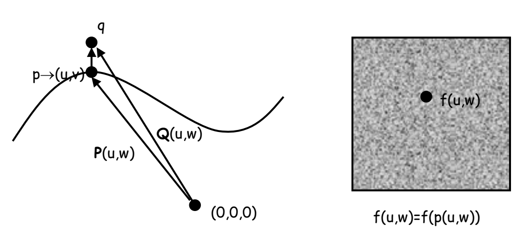
2d parameterization of smooth surface



In general $p \rightarrow (u,v)$

bump mapping: computing n_q

$\mathbf{Q}(u,w)=\mathbf{P}(u,w)+f(u,w)\mathbf{n}(u,w)$



f(u,w)=f(p(u,w))

find vectors in tangent plane

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

dQ/du = **dP/du** + [df(u,w)/du]**n**(u,w) + f(u,w)d**n**(u,w)/du

dQ/dw = **dP/dw** + [df(u,w)/dw]**n**(u,w) + f(u,w)d**n**(u,w)/dw

find vectors in tangent plane

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

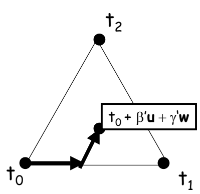
dQ/du = $\boxed{\textbf{dP/du}}$ + $\boxed{[df(u,w)/du]}$ **n**(u,w) + f(u,w) $\boxed{d\textbf{n}(u,w)/du}$

dQ/dw = $\boxed{\textbf{dP/dw}}$ + $\boxed{[df(u,w)/dw]}$ **n**(u,w) + f(u,w) $\boxed{d\textbf{n}(u,w)/dw}$

We can compute **dP/du** and **dP/dw**
for our surfaces.

triangle: parametric form

dP/du = $\lim_{\|\delta\mathbf{u}\|\to 0}$ [($t_0 + \beta'(\delta\mathbf{u} + \mathbf{u}) + \gamma\mathbf{w}$) - ($t_0 + \beta'\mathbf{u} + \gamma\mathbf{w}$)]/$\|\delta\mathbf{u}\|$ = $\beta'\mathbf{u}$



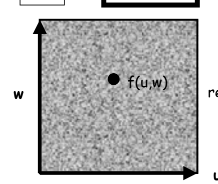
where: $\mathbf{u} = (t_1-t_0)/\|t_1-t_0\|$ is a unit vector in the t_1-t_0 direction and
$\mathbf{w} = (t_2-t_0)/\|t_2-t_0\|$ is a unit vector in the t_2-t_0 direction

find vectors in tangent plane

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

dQ/du = $\boxed{\textbf{dP/du}}$ + $\boxed{[df(u,w)/du]}$ **n**(u,w) + f(u,w) $\boxed{d\textbf{n}(u,w)/du}$

dQ/dw = $\boxed{\textbf{dP/dw}}$ + $\boxed{[df(u,w)/dw]}$ **n**(u,w) + f(u,w) $\boxed{d\textbf{n}(u,w)/dw}$



how does bump change with
respect to changes in u and w?

bump map derivative

convolution kernel

-1	0	1
-1	0	1
-1	0	1

change in u

1	1	1
0	0	0
-1	-1	-1

change in w

find vectors in tangent plane

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

dQ/du = $\boxed{\textbf{dP/du}}$ + $\boxed{[df(u,w)/du]}$ **n**(u,w) + f(u,w) $\boxed{d\textbf{n}(u,w)/du}$

dQ/dw = $\boxed{\textbf{dP/dw}}$ + $\boxed{[df(u,w)/dw]}$ **n**(u,w) + f(u,w) $\boxed{d\textbf{n}(u,w)/dw}$

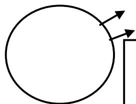
we'll call these scalars $b_u(u,w)$ and $b_w(u,w)$

find vectors in tangent plane

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

$$\mathbf{dQ/du} = \boxed{\mathbf{dP/du}} + \boxed{[df(u,w)/du]}\ \mathbf{n}(u,w) + \boxed{f(u,w)\ d\mathbf{n}(u,w)/du}$$

$$\mathbf{dQ/dw} = \boxed{\mathbf{dP/dw}} + \boxed{[df(u,w)/dw]}\ \mathbf{n}(u,w) + \boxed{f(u,w)\ d\mathbf{n}(u,w)/dw}$$



how does normal change with respect to changes in u and w

we'll ignore this because
(a) it is small,
(b) it is computationally difficult, and
(c) results look ok if we do.

find vectors in tangent plane

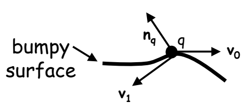
take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

$$\mathbf{dQ/du} \approx \mathbf{dP/du} + b_u(u,w)\ \mathbf{n}(u,w)$$

$$\mathbf{dQ/dw} \approx \mathbf{dP/dw} + b_w(u,w)\ \mathbf{n}(u,w)$$

we can do this!

bump mapping



computing $\mathbf{n}_q$:

1. find vectors $\mathbf{v}_0$ and $\mathbf{v}_1$ in plane tangent to bumpy surface at point q
2. $\mathbf{n}_q = (\mathbf{v}_0 \times \mathbf{v}_1)/||\mathbf{v}_0 \times \mathbf{v}_1||$

take the cross product

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

$$\mathbf{dQ/du} \approx \mathbf{dP/du} + b_u(u,w)\ \mathbf{n}(u,w)$$

$$\mathbf{dQ/dw} \approx \mathbf{dP/dw} + b_w(u,w)\ \mathbf{n}(u,w)$$

take cross product

$$\mathbf{dQ/du} \times \mathbf{dQ/dw} = \mathbf{dP/du} \times \mathbf{dP/dw} + b_u(u,w)\mathbf{dP/dw} \times \mathbf{n}(u,w) + b_w(u,w)\ \mathbf{dP/du} \times \mathbf{n}(u,w) + b_u(u,w)\ \mathbf{n}(u,w) \times b_w(u,w)\ \mathbf{n}(u,w)$$

take the cross product

take partial derivatives of **Q**(u,w)=**P**(u,w)+f(u,w)**n**(u,w) with respect to u,w

$$\mathbf{dQ/du} \approx \mathbf{dP/du} + b_u(u,w)\ \mathbf{n}(u,w)$$

$$\mathbf{dQ/dw} \approx \mathbf{dP/dw} + b_w(u,w)\ \mathbf{n}(u,w)$$

take cross product

$$\mathbf{dQ/du} \times \mathbf{dQ/dw} = \mathbf{dP/du} \times \mathbf{dP/dw} + b_u(u,w)\mathbf{dP/dw} \times \mathbf{n}(u,w) + b_w(u,w)\ \mathbf{dP/du} \times \mathbf{n}(u,w) + \boxed{b_u(u,w)\ \mathbf{n}(u,w) \times b_w(u,w)\ \mathbf{n}(u,w)}$$

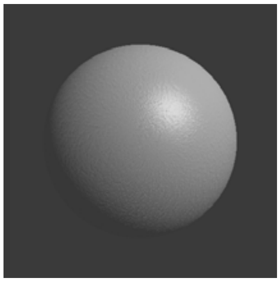
this is 0

Computation

1. Compute derivatives of surface **dP/du** and **dP/dw**
2. Compute derivatives of bump map $b_u(u,v)$ and $b_v(u,v)$
3. Take cross products and add:
   $\mathbf{dP/du} \times \mathbf{dP/dw} + b_u(u,w)\mathbf{dP/dw} \times \mathbf{n}(u,w) + b_w(u,w)\ \mathbf{dP/du} \times \mathbf{n}(u,w)$

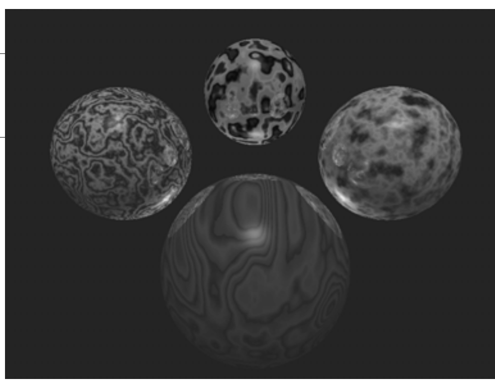
ta da!

adrian mettler,
spring 2003



procedural texture mapping

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



adrian mettler, spring 2003

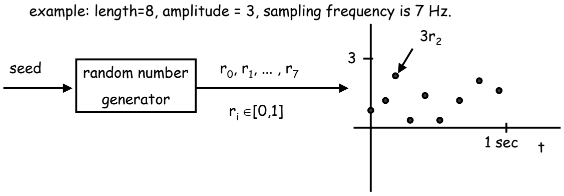
procedural textures

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

perlin noise – 1D example

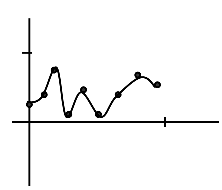
step 1: generate discrete noise function with specified length, amplitude, sampling frequency

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



perlin noise – 1D example

step 2: interpolate with smoothing

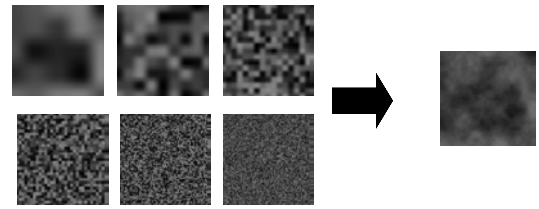


perlin noise – 1D example

step 3: repeat with various amplitudes/frequencies

step 4: add together

perlin noise – 2D example



for more info see Perlin Noise link on proj2 web site

cheap tricks

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

cheap tricks

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

Texture specifies transparency of surface

cheap tricks

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

blur based on distance from viewpoint

cheap tricks

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

See paper mentioned in assignment

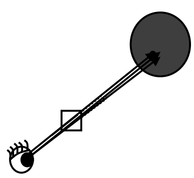
cheap tricks

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

jittering: antialiasing technique

Run rt for example

jittering: anti-aliasing technique



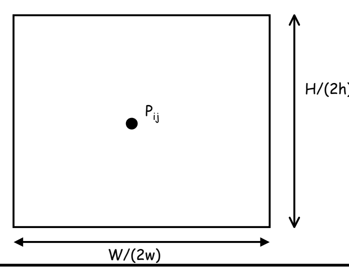
- cast several (random) rays through pixel neighborhood into scene
- for each:
  - find intersection point (if any) that is closest to eye
  - compute color at intersection
- compute average color

random ray

shoot ray i,j through P_{ij} + (dx,dy) where

dx is chosen uniformly at random over [-W/(2w), W/(2w)]

dy is chosen uniformly at random over [-H/(2h),H/(2h)]



cheap tricks

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

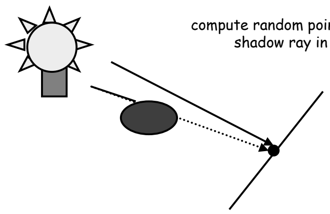
soft shadows

run rt for examples

soft shadows

In reality, we don't have point lights!!

compute random point on "light" and cast shadow ray in that direction



ray tracing

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

ray tracing complexity

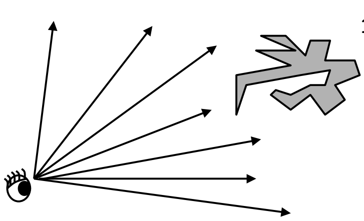
O(# of intersection tests)
= O(#pixels x # objects)

Can we reduce the number of intersection tests?

optimization

- bounding boxes
- oct-trees
- BSP-trees

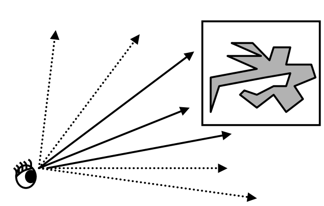
bounding boxes: intuition



10,000,00 triangles

most rays miss the object

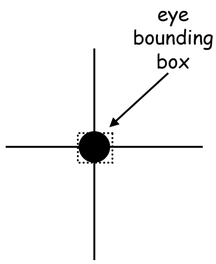
bounding boxes



1. rule out rays by simple intersection test with bounding box
2. perform exhaustive test on remaining rays

bounding boxes & hierarchical coordinates

body xfm
body description
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}

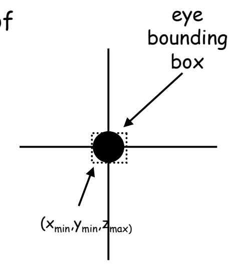


bounding boxes & hierarchical coordinates

box defined by extrema of primitive:

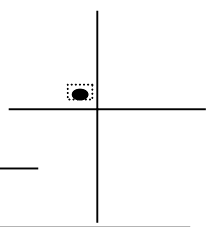
- x_{min}
- x_{max}
- y_{min}
- y_{max}
- z_{min}
- z_{max}

can you compute these
for our primitives?



bounding boxes & hierarchical coordinates

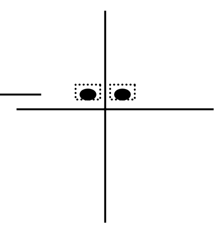
body xfm
body description
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



transformed box defined by
transformed corners $(x_{min},y_{min},z_{min})$, $(x_{min},y_{min},z_{max})$, ...

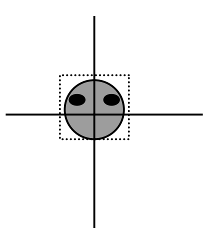
bounding boxes & hierarchical coordinates

body xfm
body description
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}

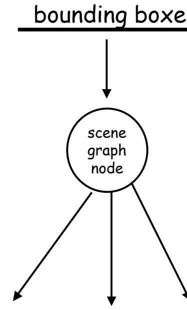


bounding boxes & hierarchical coordinates

body xfm
body description
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



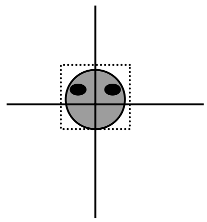
bounding boxes & hierarchical coordinates



compute extrema based on local primitive and the extrema of children's transformed bounding boxes

bounding boxes & hierarchical coordinates

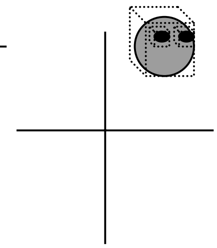
body xfm
body description
head translate wrt body
head rotate
head description ←
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



bounding box based on head and bounding boxes of yes

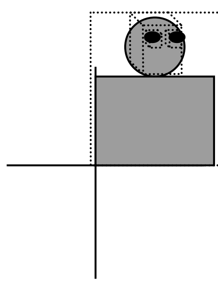
bounding boxes & hierarchical coordinates

body xfm
body description
head translate wrt body ←
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



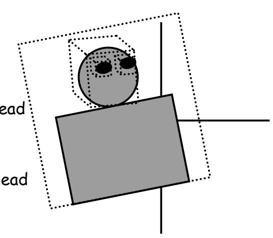
bounding boxes & hierarchical coordinates

body xfm
body description ←
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



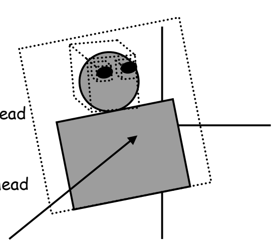
bounding boxes & hierarchical coordinates

body xfm ←
body description
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



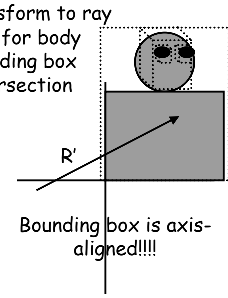
intersection

body xfm ←
body description
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}



intersection

body xfm ←Apply inverse transform to ray
body description ← Test for body bounding box intersection
head translate wrt body
head rotate
head description
{eye1 translate wrt head
eye1 scale
eye1 description}
{eye2 translate wrt head
eye2 scale
eye2 description}

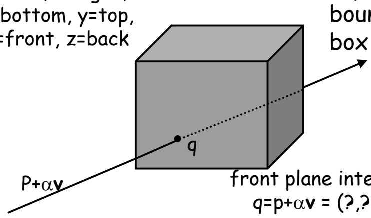


Bounding box is axis-aligned!!!!

bounding box intersection

1. find intersection of ray with each bounding plane of box

x=left, x=right,
y=bottom, y=top,
z=front, z=back



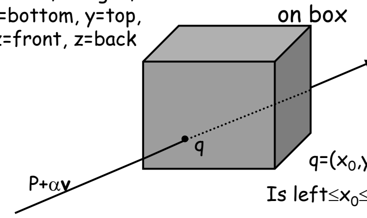
front plane intersection:
$q=p+\alpha\mathbf{v}$ = (?,?,front)

$q=(p_x+\alpha\mathbf{v}_x,\ p_y+\alpha\mathbf{v}_y,\ p_z+\alpha\mathbf{v}_z)$
where $\alpha = (\text{front}-p_z)/v_z$

bounding box intersection

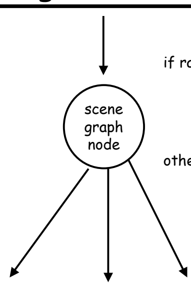
2. determine if any intersection point is on box

x=left, x=right,
y=bottom, y=top,
z=front, z=back



$q=(x_0,y_0,z_0)$
Is left$\leq x_0 \leq$right and
bottom$\leq y_0 \leq$top and
back$\leq z_0 \leq$front?

bounding box intersection test



if ray intersects with node's bounding
check for intersection with local object and
box pass ray to children
otherwise return no intersection

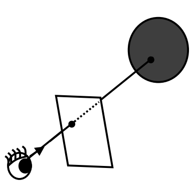
REMEMBER: bounding box encloses objects at node and all bounding boxes of children

optimization

- bounding boxes
- oct-trees
- BSP-trees

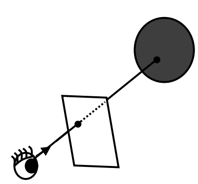
We won't cover these. Look on the web for details.

ray casting



1. cast ray through pixel into scene

ray casting



1. cast ray through pixel into scene

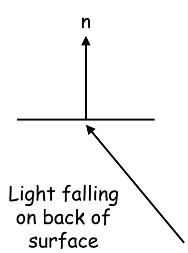
ray tracing

- ray casting
  - rays
  - intersection tests
  - intersection with scene graph
  - lighting and material properties
- **recursive ray tracing**
- cheap tricks
- optimizations

red transmission term

the red transmission term is $k_{trans}\ \Sigma\ R_{L,T}$ where
- the summation is taken over all lights L
- $R_{L.T}$ is the intensity of the red transmission of light L at the intersection point

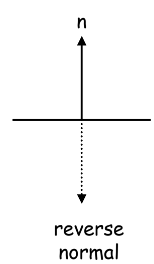
transmission



$R_{L,T} = k_{trans}\ R^{back}_{L,D}$

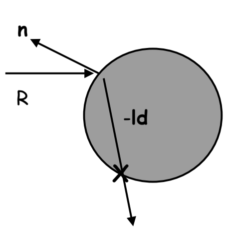
where
- $R^{back}_{L,D}$ is the luminance of diffuse reflections on the back of the surface

transmission computation



- compute $R_{L,D}$ with the reverse normal.
- scale by k_{trans}

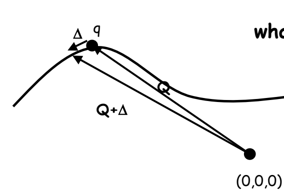
note: thick surfaces



all light falling on the back of the surface is occluded

find vectors in tangent plane

take partial derivatives of **Q** in two directions



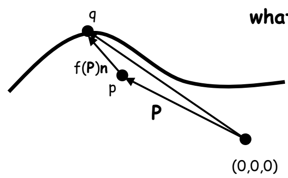
what directions?

find vectors in tangent plane

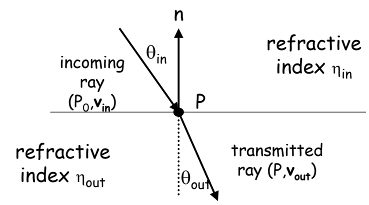
take partial derivatives of $\mathbf{Q}=\mathbf{P}+f(\mathbf{P})\mathbf{n}$ in two directions

$d\mathbf{Q}/du = d\mathbf{P}/du + d[f(\mathbf{P})\mathbf{n}]/du$

what directions?



snell's law

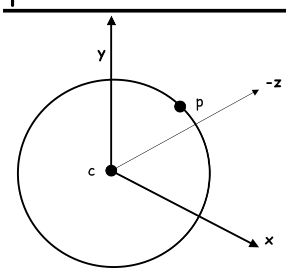


θ_{out} satisfies: $\eta_{out} \sin \theta_{out} = \eta_{in} \sin \theta_{in}$

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

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

parameterization

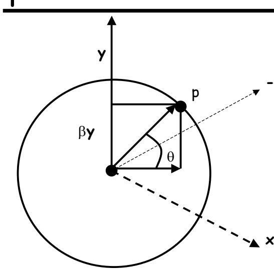


Assume

$p - c = \alpha \mathbf{x} + \beta \mathbf{y} + \gamma \mathbf{z}$

Now represent p-c in spherical coordinates

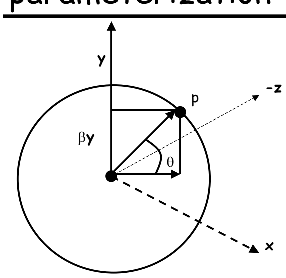
parameterization



$\beta = r \sin\theta$

$\boxed{\theta = \arcsin(\beta/r)}$

parameterization



$\beta = r \sin\theta$

$\boxed{\theta = \arcsin(\beta/r)}$

Note range of arcsin:

$\theta \in [-\pi/2, \pi/2]$

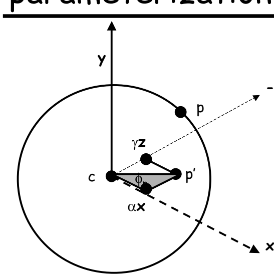
e.g.

$\beta=r$ then $\theta=\pi/2$

$\beta=0$ then $\theta=0$

$\beta=-r$ then $\theta=-\pi/2$

parameterization



$\alpha=r \cos\theta \cos \phi$

if $\cos\theta \neq 0$

$\phi = \arccos(\alpha/(r \cos\theta))$

else (undefined)

$\phi=0$

Note range of arcos:

$\phi \in [0, \pi]$

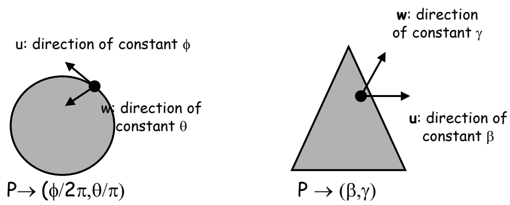
e.g. if $\theta=0$ and ...

$\alpha=r$ then $\phi=0$

$\alpha=0$ then

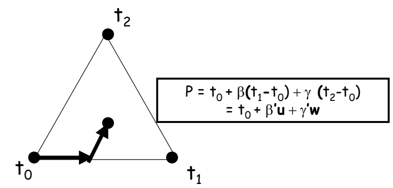
2d parameterization of smooth surface

define unit vectors u, w



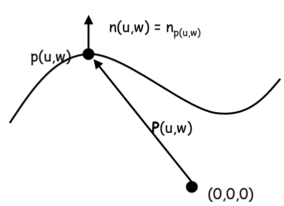
for example

$\mathbf{u} = (t_1 - t_0)/||t_1 - t_0||$



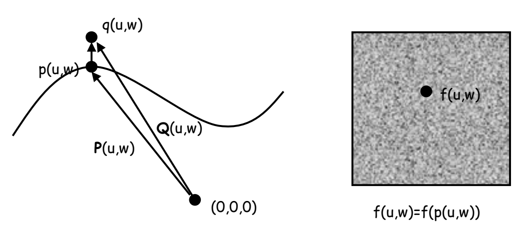
WARNING: renaming

2d parameterization of smooth surface

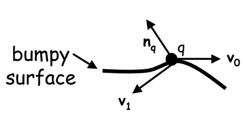


2d parameterization of bumpy surface

$\mathbf{Q}(u,w)=\mathbf{P}(u,w)+f(u,w)\mathbf{n}(u,w)$



bump mapping: computing n_q



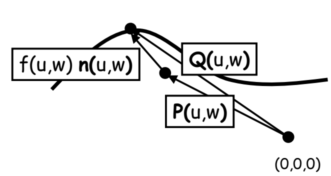
We need to do this!!

computing $\mathbf{n}_q$:

1. find vectors $\mathbf{v}_0$ and $\mathbf{v}_1$ in plane tangent to bumpy surface at point q
2. $\mathbf{n}_q = (\mathbf{v}_0 \times \mathbf{v}_1)/||\mathbf{v}_0 \times \mathbf{v}_1||$

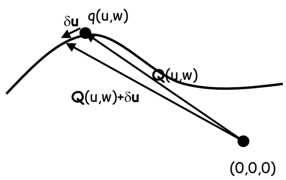
find vectors in tangent plane

take partial derivatives of $\mathbf{Q}(u,w)=\mathbf{P}(u,w)+f(u,w)\mathbf{n}(u,w)$ with respect to u,w



find vectors in tangent plane

$dQ(u,v)/du = \lim_{\delta u \to 0} Q(u,w) + \delta u$



find vectors in tangent plane

take partial derivatives of $\mathbf{Q}(u,w)=\mathbf{P}(u,w)+f(u,w)\mathbf{n}(u,w)$ with respect to u,w

$\mathbf{dQ/du} = \mathbf{dP/du} + [df(u,w)/d\mathbf{u}]\mathbf{n}(u,w) + f(u,w)d\mathbf{n}(u,w)/d\mathbf{u}$

$\mathbf{dQ/dw} = \mathbf{dP/dw} + [df(u,w)/d\mathbf{w}]\mathbf{n}(u,w) + f(u,w)d\mathbf{n}(u,w)/d\mathbf{w}$