Smallpt (continued) and the lajolla renderer

UCSD CSE 272
Advanced Image Synthesis

Tzu-Mao Li
Camera and participation

• turning on your camera helps me judge how well the materials come across

• also make me slightly more motivated

• not required

• ask questions either vocally, in chat, or in Discord!
About the scope of this course

• “Advanced” Image Synthesis

• it would be nice if you have the equivalent knowledge of CSE 168

• https://cseweb.ucsd.edu/~viscomp/classes/cse168/sp21/168.html

• or go through all of the following (short) books

https://raytracing.github.io/
Recap: what is the color of a pixel?
What is the color of a pixel?

sample the pixel filter integral
What is the color of a pixel?

find corresponding intersection point
What is the color of a pixel?

recursively query the color
What is the color of a pixel?

- If mirror, sample the mirror direction
- If diffuse, sample on the cosine weighted hemisphere
- If glass, sample reflection/refraction
What is the color of a pixel?

If emissive, include the emission
What is the color of a pixel?

- If mirror, sample the mirror direction.
- If glass, sample reflection/refraction.
- If diffuse, sample on the cosine weighted hemisphere.
Surface color is determined by a hemispherical integral

e.g., blue

\[ \iint \text{reflectance} \cdot L(\omega') | \omega' \cdot n | d\omega' \]
Surface color is determined by a hemispherical integral

e.g., blue

\[ \iiint \text{reflectance} \cdot L(\omega') | \omega' \cdot n | d\omega' \]

the cosine term is the ratio between an area on hemisphere and an area on surface
Surface color is determined by a hemispherical integral

\[ \iint L'(\omega') |\omega' \cdot n| \, d\omega' \]

\[ L' = \text{reflectance} \cdot L \]

the cosine term is the ratio between an area on hemisphere and an area on surface
Goal: sample this hemispherical integral

$$\int \int L'(\omega') |\omega' \cdot n| \, d\omega'$$

$L' = \text{reflectance} \cdot L$

the cosine term is the ratio between an area on hemisphere and an area on surface
Goal: map $\mathbf{u}$ to $\omega'$

$\mathbf{u}$ uniform distribution

$T$

$(u_1, u_2)$

$\omega'$

this mapping is a change of variable of our hemispherical integral

$$\iint L'(\omega') | \omega' \cdot n | \, d\omega' = \iint L'(\omega') \frac{| \omega' \cdot n |}{\left| \frac{\partial (u_1, u_2)}{\partial (\omega')} \right|} \, du$$
Cosine-weighted hemisphere sampling:
choose a mapping with Jacobian $1/|n \cdot \omega'|$

$\mathbf{u}$ uniform distribution

\[
\begin{align*}
\iint L'(\omega') |\omega' \cdot n| \, d\omega' &= \iint L'(\omega') \frac{|\omega' \cdot n|}{\left| \frac{du}{d\omega'} \right|} \, du = \iint L'(\omega') du
\end{align*}
\]
Malley's method:
sampling cosine-weighted hemisphere by projection

- uniformly sample a point on a disk
- project the point on the hemisphere

\[ \left| \frac{d\omega'}{du} \right| = 1/ \left| \frac{du}{d\omega'} \right| = \pi/|\omega' \cdot n| \]

geometric intuition:
the cosine term $|\omega' \cdot n|$ is the ratio between an area on hemisphere and an area on surface
so the Jacobian of the mapping is that ratio divided by the area of a unit disk.
Goal: map $\mathbf{u}$ to $\omega'$
Uniformly sampling a unit disk

- incorrect approach:
  - uniformly pick a distance $r$ from origin
  - uniformly pick an angle $\phi$

why is this wrong?
Uniformly sampling a unit disk

- incorrect approach:
  - uniformly pick a distance $r$ from origin
  - uniformly pick an angle $\phi$

why is this wrong?

“inner” circles have less area compared to “outer” circles!
Uniformly sampling a unit disk using change of variable

Disk area integral

\[ \iint rdrd\phi \]

\[ \iint du_1du_2 \]

Uniform distribution \((u_1, u_2)\)
Strategy: one dimension at a time
(deal with r first)

Disk area integral

\[ \int \int r \, dr \, d\phi = 2\pi \int r \, dr \]

\[ = 2\pi \int \frac{r \, dr}{du_1} \, du_1 \]

goal: find a transformation \( r = T(u_1) \) s.t. \( \frac{du_1}{dr} \propto r \)
Strategy: one dimension at a time
(deal with r first)

disk area integral

\[ \iint r\,dr\,d\phi = 2\pi \int r\,dr \]

\[ = 2\pi \int \frac{r}{du_1} \, du_1 \]

Goal: find a transformation \( u_1 = T_1^{-1}(r) \) s.t. \( \frac{du_1}{dr} \propto r \)
Strategy: one dimension at a time
(deal with $r$ first)

Disk area integral

$$\iint r dr d\phi = 2\pi \int rd\phi$$

$$= 2\pi \int \frac{r}{du_1} du_1$$

Goal: find a transformation $u_1 = T^{-1}(r)$ s.t. $\frac{du_1}{dr} \propto r$

$$u_1 \propto \int_0^r r' dr' = \frac{1}{2} r^2$$

$$u_1 = \frac{A}{2} r^2$$
Strategy: one dimension at a time  
(deal with r first)

disk area integral

\[ \iint r dr d\phi = 2\pi \int r dr \]

\[ = 2\pi \int \frac{r}{\frac{du_1}{dr}} du_1 \]

Goal: find a transformation \( u_1 = T_1^{-1}(r) \) s.t. \( \frac{du_1}{dr} \propto r \)

\[ u_1 \propto \int_0^r r' dr' = \frac{1}{2} r^2 \quad u_1 = \frac{A}{2} r^2 \]

add constraints \( T_1^{-1}(0) = 0, T_1^{-1}(1) = 1 \)  
(so that \( u_1 \in [0,1] \))

\[ A = 2 \]
Strategy: one dimension at a time
(deal with r first)

\[ \int \int r \, dr \, d\phi = 2\pi \int r \, dr \]

\[ = 2\pi \int \frac{r}{\frac{du_1}{dr}} \, du_1 \]

Goal: find a transformation \( u_1 = T_1^{-1}(r) \) s.t. \( \frac{du_1}{dr} \propto r \)

\( u_1 = r^2 \)
Strategy: one dimension at a time
(deal with r first)

disk area integral

\[ \iint r \, dr \, d\phi = 2\pi \int r \, dr \]

\[ = 2\pi \int \frac{r}{du_1} \, du_1 \]

Goal: find a transformation \( u_1 = T_1^{-1}(r) \) s.t. \( \frac{du_1}{dr} \propto r \)

\( u_1 = r^2 \)

\( r = \sqrt{u_1} \)
Strategy: one dimension at a time
(deal with $\phi$ next)

disk area integral

\[ \iint r \, dr \, d\phi = 2\pi \int r \, dr \]

\[ = 2\pi \int \frac{r}{\frac{du_1}{dr}} \, du_1 \]

Goal: find a transformation $u_1 = T_1^{-1}(r)$ s.t. $\frac{du_1}{dr} \propto r$

\[ u_1 = r^2 \]
\[ r = \sqrt{u_1} \]

\[ \phi = 2\pi u_2 \]
Mapping a square to a uniform disk

\begin{align*}
r &= \sqrt{u_1} \\
\phi &= 2\pi u_2
\end{align*}
Malley’s method: sampling cosine-weighted hemisphere by projection

- uniformly sample a point on a disk
- project the point on the hemisphere

\[ r = \sqrt{u_1} \]
\[ \phi = 2\pi u_2 \]

\[ \cos \theta = \sqrt{1 - r^2} \]

exercise: compute the Jacobian and show the correctness
Cosine-weighted hemisphere sampling allows us to approximate integrals

\[
\iint L'(\omega') | \omega' \cdot n | \, d\omega'
\]

\[
\approx \frac{1}{N} \sum_{i=1}^{N} \frac{L'(\omega'(u_i)) | \omega'(u_i) \cdot n |}{p(\omega'(u_i))}
\]

\[
p(\omega') = \frac{|\omega' \cdot n|}{\pi}
\]

probability density function = Jacobian
Vec radiance(const Ray &r, int depth, unsigned short *Xi) {

if (obj.refl == DIFF) {
    // Ideal DIFFUSE reflection
    // construct a local orthonormal basis
    Vec w=nl;
    Vec u=((fabs(w.x)>.1?Vec(0,1,0):Vec(1,0,0))%w).norm();
    Vec v=w%u; // cross product
    // importance sample |n dot omega|
    double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
    Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
    return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
}
What is the color of a pixel?

- If mirror, sample the mirror direction.
- If glass, sample reflection/refraction.
- If diffuse, sample on the cosine weighted hemisphere.
Smallpt code: mirror reflection

1. } else if (obj.refl == SPEC) // Ideal SPECULAR reflection
2. Ray refl_ray = Ray(x, r.d - n*2*n.dot(r.d));
3. return obj.e + f.mult(radiance(refl_ray, depth, Xi));
What is the color of a pixel?

- If mirror, sample the mirror direction.
- If glass, sample reflection/refraction.
- If diffuse, sample on the cosine weighted hemisphere.
Glass refraction

\[ \frac{\sin \theta'}{\sin \theta} = \frac{\eta}{\eta'} \]
Glass refraction

\[ \frac{\sin \theta'}{\sin \theta} = \frac{\eta}{\eta'} \]

how much light reflected/refracted?
Things become more reflective and less refractive at "grazing angles".

more reflective

less reflective
Fresnel equation predicts the ratio of reflection/refraction

- It's an electromagnetic phenomenon

\[ F = R_s^2 + R_p^2 \]
\[ R_s = \frac{\tan(\theta - \theta')}{\tan(\theta + \theta')} \]
\[ R_p = -\frac{\sin(\theta - \theta')}{\sin(\theta + \theta')} \]

From energy conservation & light wave oscillation direction

Graphics people use Schlick’s approximation

\begin{align*}
F &\approx F_0 + (1 - F_0)(1 - \cos \theta)^5 \\
F_0 &= \left( \frac{\eta - \eta'}{\eta + \eta'} \right)^2
\end{align*}

small $\cos \theta$, $F \sim 1$

large $\cos \theta$, $F \sim F_0$
1. Ray `reflRay(x, r.d-n*2*n.dot(r.d));`
2. `bool into = n.dot(nl)>0;` // Ray from outside going in?
3. `double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;`
4. `if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0) // Total internal reflection`
5. `return obj.e + f.mult(radiance(reflRay,depth,Xi));`
6. `Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t))).norm();`
7. `double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));`
8. `double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re;`
9. `double P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);`
10. `return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ?

```
radiance(reflRay,depth,Xi)*RP:
radiance(Ray(x,tdir),depth,Xi)*TP) :
radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
```

"When in doubt, use 1.5", Peter Shirley

https://psgraphics.blogspot.com/2020/03/fresnel-equations-schlick-approximation.html
Terminate condition for recursion

when do we stop?
Terminate condition for recursion

when do we stop?

probabilistically terminate the path
("Russian roulette")

```c
if (++depth>5)
if (erand48(Xi)<p) f=f*(1/p);
else return obj.e;
```
Voila!

8 samples per pixel
40 samples per pixel
200 samples per pixel
1000 samples per pixel
Recap: what is the color of a pixel?
What is the color of a pixel?

sample the pixel filter integral
What is the color of a pixel?

find corresponding intersection point
What is the color of a pixel?

recursively query the color
What is the color of a pixel?

- If mirror, sample the mirror direction.
- If diffuse, sample on the cosine weighted hemisphere.
- If glass, sample reflection/refraction.
What is the color of a pixel?

if emissive, include the emission
From smallpt to lajolla
Smallpt: sphere geometry only

1. Sphere spheres[] = {
   //Scene: radius, position, emission, color, material
   2. Sphere(16.5, Vec(27, 16.5, 47)), Vec(), Vec(1,1,1)*.999, SPEC), //Mirr
   3. Sphere(16.5, Vec(73, 16.5, 78)), Vec(), Vec(1,1,1)*.999, REFR), //Glas
   4. Sphere(600, Vec(50, 681.6-.27, 81.6), Vec(12,12,12), Vec(), DIFF) //Lite
11. };

   1. Sphere spheres[] = {
   //Scene: radius, position, emission, color, material
   2. Sphere(1e5, Vec(1e5+1,40.8,81.6), Vec(), Vec(.75,.25,.25),DIFF), //Left
   3. Sphere(1e5, Vec(-1e5+99,40.8,81.6), Vec(), Vec(.25,.25,.75),DIFF), //Rght
   4. Sphere(1e5, Vec(50,40.8,1e5), Vec(), Vec(.75,.75,.75),DIFF), //Back
   5. Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(), Vec(), DIFF), //Frnt
   6. Sphere(1e5, Vec(50, 1e5, 81.6), Vec(), Vec(.75,.75,.75),DIFF), //Botm
   7. Sphere(1e5, Vec(50,-1e5+81.6,81.6), Vec(), Vec(.75,.75,.75),DIFF), //Top
   8. Sphere(16.5, Vec(27,16.5,47), Vec(), Vec(1,1,1)*.999, SPEC), //Mirr
   9. Sphere(16.5, Vec(73,16.5,78), Vec(), Vec(1,1,1)*.999, REFR), //Glas
   10. Sphere(600, Vec(50,681.6-.27,81.6), Vec(12,12,12), Vec(), DIFF) //Lite
11. };

   1. Sphere spheres[] = {
   //Scene: radius, position, emission, color, material
   2. Sphere(1e5, Vec(1e5+1,40.8,81.6), Vec(), Vec(.75,.25,.25),DIFF), //Left
   3. Sphere(1e5, Vec(-1e5+99,40.8,81.6), Vec(), Vec(.25,.25,.75),DIFF), //Rght
   4. Sphere(1e5, Vec(50,40.8,1e5), Vec(), Vec(.75,.75,.75),DIFF), //Back
   5. Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(), Vec(), DIFF), //Frnt
   6. Sphere(1e5, Vec(50, 1e5, 81.6), Vec(), Vec(.75,.75,.75),DIFF), //Botm
   7. Sphere(1e5, Vec(50,-1e5+81.6,81.6), Vec(), Vec(.75,.75,.75),DIFF), //Top
   8. Sphere(16.5, Vec(27,16.5,47), Vec(), Vec(1,1,1)*.999, SPEC), //Mirr
   9. Sphere(16.5, Vec(73,16.5,78), Vec(), Vec(1,1,1)*.999, REFR), //Glas
   10. Sphere(600, Vec(50,681.6-.27,81.6), Vec(12,12,12), Vec(), DIFF) //Lite
11. };

   1. Sphere spheres[] = {
   //Scene: radius, position, emission, color, material
   2. Sphere(1e5, Vec(1e5+1,40.8,81.6), Vec(), Vec(.75,.25,.25),DIFF), //Left
   3. Sphere(1e5, Vec(-1e5+99,40.8,81.6), Vec(), Vec(.25,.25,.75),DIFF), //Rght
   4. Sphere(1e5, Vec(50,40.8,1e5), Vec(), Vec(.75,.75,.75),DIFF), //Back
   5. Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(), Vec(), DIFF), //Frnt
   6. Sphere(1e5, Vec(50, 1e5, 81.6), Vec(), Vec(.75,.75,.75),DIFF), //Botm
   7. Sphere(1e5, Vec(50,-1e5+81.6,81.6), Vec(), Vec(.75,.75,.75),DIFF), //Top
   8. Sphere(16.5, Vec(27,16.5,47), Vec(), Vec(1,1,1)*.999, SPEC), //Mirr
   9. Sphere(16.5, Vec(73,16.5,78), Vec(), Vec(1,1,1)*.999, REFR), //Glas
   10. Sphere(600, Vec(50,681.6-.27,81.6), Vec(12,12,12), Vec(), DIFF) //Lite
11. };
More commonly used geometry primitive: triangle mesh
Ray-triangle intersection

\[ \mathbf{x} = \mathbf{o} + t \cdot \mathbf{d} \]

\[ \mathbf{x} = (1 - b_1 - b_2) \mathbf{P}_0 + b_1 \mathbf{P}_1 + b_2 \mathbf{P}_2 \]

https://en.wikipedia.org/wiki/Barycentric_coordinate_system
Ray-triangle intersection

\[
o + t \cdot d = (1 - b_1 - b_2)P_0 + b_1P_1 + b_2P_2
\]

\[
o_x + t \cdot d_x = (1 - b_1 - b_2)P_{0x} + b_1P_{1x} + b_2P_{2x}
\]

\[
o_y + t \cdot d_y = (1 - b_1 - b_2)P_{0y} + b_1P_{1y} + b_2P_{2y}
\]

\[
o_z + t \cdot d_z = (1 - b_1 - b_2)P_{0z} + b_1P_{1z} + b_2P_{2z}
\]

3 unknowns (t, b1, b2), 3 linear equations

“Moller-Trumbore algorithm”
Lajolla supports triangle meshes and spheres

A triangle mesh representing a quad:

positions = {{-1, -1, 0},
             { 1, -1, 0},
             {-1,  1, 0},
             { 1,  1, 0}};
indices = {{0, 1, 2},
           {2, 1, 3}}

https://github.com/BachiLi/lajolla_public/blob/main/src/shape.h
Smallpt: hope to hit light with directional sampling

- If mirror, sample the mirror direction
- If glass, sample reflection/refraction
- If diffuse, sample on the cosine weighted hemisphere
Next event estimation

- in addition to cosine-weighted hemisphere sampling, also sample a point on light
Next event estimation is also a change of variable

Focus on the rays that hit the light source

\[ \iint L_e'(\omega') | \omega' \cdot n | \, d\omega' \]
Next event estimation is also a change of variable

\[ \iint L_e'(\omega') | \omega' \cdot n | d\omega' \]

focus on the rays that hit the light source

\[ \iint L_e'(\omega'(p')) | \omega'(p') \cdot n_p | \frac{| \omega'(p') \cdot n_{p'} |}{\| p - p' \|^2} \text{visible}(p, p') dp' \]

\[ \text{the Jacobian (often called “geometry term”) } \]
Handling multiple bounces
Handling multiple bounces
Handling multiple bounces
Triangle-mesh light sampling:

1. pick a light based on their intensities
2. pick a triangle based on its area
3. pick a point on the triangle

see https://cseweb.ucsd.edu/~tzli/cse272/lectures/triangle_sampling.pdf for notes on triangle sampling
Sampling a discrete distribution

- treat it as a piecewise constant function

\[
\text{find } u = T^{-1}(x) \text{ s.t. } \frac{du}{dx} = f(x)
\]

https://www.pbr-book.org/3rd-2018/Monte_Carlo_Integration/Sampling_Random_Variables#x1-Example:Piecewise-Constant1DFunctions
https://github.com/BachiLi/lajolla_public/blob/main/src/table_dist.cpp
Next event estimation is good at small lights

next event estimation
(64 samples per pixel)

cosine weighted hemisphere
(64 samples per pixel)
When the point is close to the light, cosine-weighted hemisphere sampling is better

next event estimation

Cosine-weighted hemisphere sampling

“Robust Monte Carlo Methods for Light Transport Simulation”, Eric Veach
Multiple importance sampling: combining next event estimation and hemisphere sampling

\[ \iint L'(\omega') | \omega' \cdot n | \, d\omega' \approx \frac{1}{N} \sum_{i=1}^{N} \frac{L'(\omega_i') | \omega_i' \cdot n |}{p_{\text{nee}}(\omega_i')} \]

\[ \approx \frac{1}{N} \sum_{i=1}^{N} \frac{L'(\omega_i') | \omega_i' \cdot n |}{p_{\text{hemi}}(\omega_i')} \]
Multiple importance sampling: combining next event estimation and hemisphere sampling

- idea: assign higher weight when \( p \) is high

\[
\frac{1}{N} \left( \sum_{i=1}^{N} w_i \frac{L'(\omega'_i) | \omega'_i \cdot n |}{p_{\text{nee}}(\omega'_i)} + \sum_{j=1}^{N} w_j \frac{L'(\omega'_j) | \omega'_j \cdot n |}{p_{\text{hemi}}(\omega'_j)} \right)
\]
Multiple importance sampling:
combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

\[
\frac{1}{N} \left( \sum_{i=1}^{N} w^n_i \frac{L'(\omega'_i) | \omega'_i \cdot n |}{p_{\text{nee}}(\omega'_i)} + \sum_{j=1}^{N} w^h_j \frac{L'(\omega'_j) | \omega'_j \cdot n |}{p_{\text{hemi}}(\omega'_j)} \right)
\]

\[
w^n_i = \frac{p_{\text{nee}}(\omega_i)}{p_{\text{nee}}(\omega_i) + p_{\text{hemi}}(\omega_i)}
\]
Multiple importance sampling: combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

\[
\frac{1}{N} \left( \sum_{i=1}^{N} w_i^n \frac{L'(\omega'_i) \mid \omega'_i \cdot n \mid}{p_{\text{nee}}(\omega'_i)} + \sum_{j=1}^{N} w_j^h \frac{L'(\omega'_j) \mid \omega'_j \cdot n \mid}{p_{\text{hemi}}(\omega'_j)} \right)
\]

\[
w_i^n = \frac{p_{\text{nee}}(\omega_i)}{p_{\text{nee}}(\omega_i) + p_{\text{hemi}}(\omega_i)} \quad w_j^h = \frac{p_{\text{hemi}}(\omega_j)}{p_{\text{nee}}(\omega_j) + p_{\text{hemi}}(\omega_j)}
\]
Multiple importance sampling:
combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

$$\frac{1}{N} \left( \sum_{i=1}^{N} w_i^n \frac{L'(\omega_i') | \omega_i' \cdot n |}{p_{\text{nee}}(\omega_i')} + \sum_{j=1}^{N} w_j^h \frac{L'(\omega_j') | \omega_j' \cdot n |}{p_{\text{hemi}}(\omega_j')} \right)$$

$$w_i^n = \frac{p_{\text{nee}}(\omega_i)}{p_{\text{nee}}(\omega_i) + p_{\text{hemi}}(\omega_i)}$$

$$w_j^h = \frac{p_{\text{hemi}}(\omega_j)}{p_{\text{nee}}(\omega_j) + p_{\text{hemi}}(\omega_j)}$$

these two are different!!

see https://github.com/BachiLi/lajolla_public/blob/main/src/path_tracing.h for details on matching the units/measures. It also uses a slightly different weight.
MIS combines the best of both worlds

next event estimation

cosine-weighted hemisphere sampling

multiple importance sampling

“Robust Monte Carlo Methods for Light Transport Simulation”, Eric Veach
Environment maps

- an infinitely far area light source represented as an image
Environment maps

- an infinitely far area light source represented as an image
Environment maps sampling: treat it as a big discrete 2D table

- first sample a row, then sample a column

images from https://www.pbr-book.org/3ed-2018/Monte_Carlo_Integration/2D_Sampling_with_Multidimensional_Transformations
see https://www.pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Light_Sources#InfiniteAreaLights for more details
Smallpt: constant color across the surface

1. Sphere spheres[] = {
2. Sphere(1e5, Vec(1e5+1,40.8,81.6), Vec(), Vec(.75,.25,.25),DIFF), // Left
3. Sphere(1e5, Vec(-1e5+99,40.8,81.6), Vec(), Vec(.25,.25,.75),DIFF), // Right
4. Sphere(1e5, Vec(50,40.8,1e5), Vec(), Vec(.75,.75,.75),DIFF), // Back
5. Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(), Vec(), DIFF), // Front
6. Sphere(1e5, Vec(50,1e5,81.6), Vec(), Vec(.75,.75,.75),DIFF), // Bottom
7. Sphere(1e5, Vec(50,-1e5+81.6,81.6), Vec(), Vec(.75,.75,.75),DIFF), // Top
8. Sphere(16.5, Vec(27,16.5,47), Vec(), Vec(1,1,1)*.999, SPEC), // Mirror
9. Sphere(16.5, Vec(73,16.5,78), Vec(), Vec(1,1,1)*.999, REFR), // Glass
10. Sphere(600, Vec(50,681.6-.27,81.6), Vec(12,12,12), Vec(), DIFF) // Lite
11. };
Real-world surfaces are colorful!


photo from K.C. Alfred
An option: assign a color to each triangle

• pros
  • simple
  • easy to edit (just paint on triangles)
• cons
  • couples geometric complexity with color complexity
  • hard to filter
  • more on this later
In practice: texture mapping

- assign a “UV” 2D vector to each point on the surface

![Image of texture mapping](texture_from_pbtr-v2_pbtr.org/scenes-v2)
UV mapping

• “unwrap” a surface and map it to a 2D square
• automatic UV mapping is an active research area

http://staff.ustc.edu.cn/~fuxm/projects/Peeling/index.html
Obtain UV by interpolating values from vertices

\[ uv = (1 - b_1 - b_2)uv_0 + b_1 uv_1 + b_2 uv_2 \]
Texture mapping: pros and cons

• pros
  • different sampling rates for geometry and color
  • much easier to filter
• cons
  • uv mapping is hard
Texture mapping: pros and cons

• pros
  • different sampling rates for geometry and color
  • much easier to filter

• cons
  • uv mapping is hard
A pixel can cover a large region in the texture
Fail to account for all texels in the region can lead to noise/aliasing.


- no filtering
- with filtering
Goal: average all texture values inside the region
The filtering region is determined by the mapping between image space and texture space.

The mapping is non-linear in general.
We can approximate the local mapping using first-order Taylor expansion

\[
u, v \approx T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}
\]
We can approximate the local mapping using first-order Taylor expansion

\[ u, v = T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \]

approximate local regions using an ellipse

Figure 8. Contours of elliptical paraboloid \( Q \) and box around \( Q = F \). Dots are centers of texture space pixels.

"Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter", Greene and Heckbert 1986
Goal: average all texture values inside the region
Downsample the texture for fast average

• usually called “mipmapping”
Elliptical weighted averaging algorithm
Elliptical weighted averaging algorithm

1. approximate the ellipse with circles
Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel
Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

2. find two appropriate mipmap levels s.t. each circle maps to \( \sim 1 \) texel

3. linearly interpolate between pixels and mipmap levels
Modern video games/renderers use EWA filtering!

How do we obtain the derivatives?

\[ u, v = T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \]
The mapping $T$ is the ray casting function

- so we can simply apply chain rule and differentiate ray casting

$$u, v = T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$
The mapping $T$ is the ray casting function

- so we can simply apply chain rule and differentiate ray casting

Tracing Ray Differentials

Homan Igehy
Computer Science Department
Stanford University

Figure 1: A Ray Differential. The diagram above illustrates the positions and directions of a ray and a differentially offset ray after a reflection. The difference between these positions and directions represents a ray differential.
Lajolla uses a heuristic method to approximate the ellipse with a circle

- see homework 0 for more details and the reasoning

initial spread = (pixel size) / 4

more about texture filtering next Monday!

https://github.com/BachiLi/lajolla_public/blob/main/src/texture.h
Texture mapping: pros and cons

• pros
  • different sampling rates for geometry and color
  • much easier to filter

• cons
  • uv mapping is hard
In movie production: use UV generated by mesh subdivision

- will talk more about this in the later lectures

Ptex: Per-Face Texture Mapping for Production Rendering

Brent Burley\textsuperscript{1} and Dylan Lacewell\textsuperscript{1,2}

\textsuperscript{1}Walt Disney Animation Studios
\textsuperscript{2}University of Utah

\textbf{Figure 4:} a) Portion of a control mesh showing intrinsic faceids and edgeids. b) Corresponding limit surface showing continuous isolines across faces.
Shading normals

- triangle meshes are planar approximation of a smooth surface: can look faceted.
Trick: assign a normal per vertex, then interpolate

\[ n = \text{normalize} \left( (1 - b_1 - b_2)n_0 + b_1 n_1 + b_2 n_2 \right) \]
w/o per-vertex normal  

w/ per-vertex normal
How to get vertex normal?

- weighted average of normals of nearby faces
Alternative: normal mapping
Discrepancies between shading normal and real geometry can lead to artifacts.

From “Hacking the Shadow Terminator”, Johannes Hanika
Discrepancies between shading normal and real geometry can lead to artifacts

Lajolla doesn’t implement this!

From “Hacking the Shadow Terminator”, Johannes Hanika
For shading, we need a local coordinate basis

\[
\frac{dp}{du} - n(n \cdot \frac{dp}{du})
\]

in lajolla, stored in `PathVertex::shading_frame`
Next time:

**bidirectional scattering distribution function**

\[
L(p, \omega) = L_e(p, \omega) + \int f_p(\omega, \omega') L(p', -\omega') |n_p \cdot \omega'| d\omega'
\]