Datorgrafik DT3025

Martin Magnusson
January 14, 2016

Jourhavande lärare Martin Magnusson, telefon 073 667 2258.

Tools No tools (books or calculators) are allowed.

There are three types of questions on this exam, targetted for the goals that correspond to a passing grade (3) and a pass with distinction (grades 4–5). There are three questions for each topic. To pass with grade 3 (G), you need 12 out of 16 points (75%) from the the {3} questions. To pass with grade 4 (VG), you additionally need 75% from the {4} questions. To pass with grade 5, you additionally need 75% from the {5} questions.

You may answer in Swedish or English.

Martin will come by at around 3 o'clock, to answer questions in case something needs to be clarified.

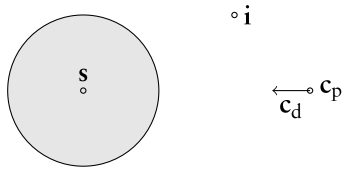
Good luck!

Question 1

The reflectance of materials can be modelled in a large number of ways. One particular model that has been well used historically (since the '70s), and is still quite common, is the Phong model.

4 points {3} Explain (in words) the four components used in the traditional Phong model and how they differ between black rubber and glossy plastic.

4 points {4} Given a sphere with material components $C_e = (0,0,0)$, $C_a = (0,0,0)$, $C_d = (0.5, 0.5, 0.5)$, $C_s = (0.5, 0.5, 0.5)$, $f = 1$, radius $r = 1$ and centre point $\mathbf{s} = (0,0,0)$, a camera at position $\mathbf{c}_p = (3,0,0)$ and direction $\mathbf{c}_d = (-1,0,0)$, and an isotropic white point light at position $\mathbf{i} = (2,0,1)$, compute the colour of the image's centre pixel using the traditional Phong model.



4 points {5} Discuss phenomenological, measure-based, and physically-based models for light scattering and relate to the traditional Phong model.

Solution

{3} Half a point each for listing diffuse, specular, ambient, and emissive. It's also fine to list specular gain and exponent separately.

One point each for describing sensible parameters for rubber and plastic, respectively.

Black rubber should have

- no emission,

- an ambient colour of $C_\mathrm{a} = (0,0,0)$ if used in a rendering method that includes indirect illumination or $C_\mathrm{a} \approx (0.1, 0.1, 0.1)$ if used in the "traditional" sense,
- a diffuse colour close to $C_\mathrm{d} = (0,0,0)$,
- a specular colour also close to $(0,0,0)$ (or, alternatively, $k_\mathrm{s} \cdot (1,1,1)$ with k_s close to zero),[1] and
- the specular exponent should be around $f = 1$, or at least $f < 10$. (Although if $k_\mathrm{s} = 0$, the value of f doesn't matter.)

Glossy plastic should have

- no emission,
- an ambient colour of $C_\mathrm{a} = (0,0,0)$ if used in a rendering method that includes indirect illumination or $C_\mathrm{a} \approx (0.1, 0.1, 0.1)$ if used in the "traditional" sense,
- a diffuse colour (r, g, b) depending on the colour of the plastic,
- a specular colour of $k_\mathrm{s} \cdot (1,1,1)$ with $k_\mathrm{s} \leqslant 1$ and substantially larger than zero,
- and a specular exponent f somewhere between 10 and 100.

{4} The centre pixel will be on the first surface hit by a ray starting at $\mathbf{c}_\mathrm{p}$, in direction $\mathbf{c}_\mathrm{d}$. This is the point $\mathbf{h} = (1,0,0)$ on the sphere. To compute the colour using the Phong model, we need the normal vector $\mathbf{n}$ at this point, the view vector $\mathbf{v}$, the direction $\boldsymbol{\omega}_\mathrm{i}$ from $\mathbf{h}$ to $\mathbf{i}$, and the mirror direction $\boldsymbol{\omega}_\mathrm{r}$. The colour is computed as

$$C = C_\mathrm{e} + C_\mathrm{a} + C_\mathrm{d}(\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n}) + C_\mathrm{s}(\mathbf{v} \cdot \boldsymbol{\omega}_\mathrm{r})^f. \quad (1)$$

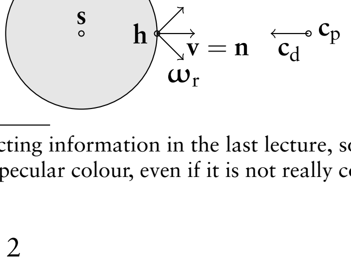
Since we have white light ($I = (1,1,1)$), we can omit the I colour terms from the equation. For a unit sphere, the normal vector is the same as the position vector of the surface point, so $\mathbf{n} = (1,0,0)$. The direction to the light is $\mathbf{i} - \mathbf{h} = (2,0,1) - (1,0,0) = (1,0,1)$. Normalising the vector gives $\boldsymbol{\omega}_\mathrm{i} = (1,0,1) / \left\|(1,0,1)\right\| = (1/\sqrt{2}, 0, 1/\sqrt{2})$. The direction to the camera is $\mathbf{c}_\mathrm{p} - \mathbf{h} = (3,0,0) - (1,0,0)$, so the normalised view vector is $\mathbf{v} = (1,0,0)$. Because the normal is parallel to the x axis, it is particularly easy to compute the mirror direction $\boldsymbol{\omega}_\mathrm{r} = (1/\sqrt{2}, 0, -1/\sqrt{2})$.

Plugging these numbers into Equation 1, we get

$$C = C_\mathrm{e} + C_\mathrm{a} + C_\mathrm{d}(1/\sqrt{2}) + C_\mathrm{s}(1/\sqrt{2})^f. \quad (2)$$

Since all the colours of the sphere's materials have equal red, green, and blue components, the colour of the pixel will be $C = (k, k, k)$, where

$$k = 0 + 0 + 0.5(1/\sqrt{2}) + 0.5(1/\sqrt{2}) = 1/\sqrt{2}. \quad (3)$$



[1] I might have been giving conflicting information in the last lecture, so I have given points also for those who answered with a bright specular colour, even if it is not really correct.

{5} One point each for a description of what phenomenological, measure-based, and physically-based models are, and one point for describing where the Phong model fits.

Phenomenological: ad-hoc, trying to device something that looks right. Measure-based: using gonireflectometer to collect data, fit a model. Works for a single material. Physically-based: try to model (some aspects of) underlying physics. (Examples [assuming geometric optics]: Cook-Torrance, Oren-Nayar) Works, hopefully, for a range of materials. Phong is phenomenological.

Question 2

The rendering equation can be written as

$$L(P,\omega_o) = \underbrace{L^e(P,\omega_o)}_{A} + \int_{\omega_i \in S^2(P)} \underbrace{L(P,-\omega_i)}_{B} \underbrace{f_s(P,\omega_i,\omega_o)}_{C} \underbrace{(\omega_i \cdot \mathbf{n}_P)}_{D} \, d\omega_i.$$

4 points {3} Describe what the terms A–D mean.

4 points {4} Describe how the rendering equation is implemented in ray tracing and path tracing, respectively.

4 points {5} Describe why the rendering equation is difficult to solve in general. List two reasons, and mention why those are problematic.

Solution

{3} One point each for

A: light emitted from P in direction ω_o,

B: incoming light to P from direction ω_i,

C: how much of 1the light that arrives at P from ω_i scatters in direction ω_o,

D: the cosine of the angle between the incoming light and the normal at P (assuming normalised vectors).[2]

2 points {4} Path tracing rather computes the integral "depth first," by shooting a single secondary ray per recursion, where a ray hits a surface. The rendering progresses by iterating over all pixels multiple times, each time sampling a new ω_i direction in order to better estimate the integral, but can be stopped at any moment. In that sense, it is more of an "any-time" algorithm than basic ray tracing. You quickly get a noisy result, which gets better the longer you wait.

2 points In basic ray tracing, a single ray is shot per pixel, and the integral is estimated by a fixed (and usually small) number of samples: one per light source with ω_i = the direction to the light source, optionally (if the material has mirror reflection) a reflected recursive ray, and optionally (if the material has translucency) a transmitted recursive ray. This gives a low-variance result (in fact, it is purely deterministic), but with a bias compared to the true solution (which integrates over all directions). In "distribution ray tracing," at any recursion step, multiple rays may be emitted, but this can quickly lead to a combinatorial explosion of rays ("breadth-first").

[2]This should actually be $|\omega_i \cdot \mathbf{n}_P|$ since this version of the rendering equation includes light transmitted from within the object — integrating over $\omega_i \in S^2(P)$.

{5} One point each for something that covers the following four items.

- It is an integral equation, where the function $L(P, \omega_o)$ that we want to find is also on the right hand side of the equation, inside an integral. (We need to know the function L for computing the function $L\ldots$)
- The integral cannot usually be solved analytically.
- Because it involves shadowing from other obstacles, etc, so we need to evaluate it numerically instead, using discrete samples. (In practice, most modern methods use randomised samples: Monte Carlo integration.)
- Choosing the samples wisely, to get low variance (little noise) but still get an image decently fast, is nontrivial. (Importance sampling, multiple importance sampling, etc.)
- Not only reflections from surfaces contribute to the incoming light. There could also be light scattered in participating media.
- Subsurface scattering is not (at lest not explicitly) included in this version of the rendering equation. One would also need a second integral, integrating over output *points* (not just out directions), to cover that.

Question 3

Linear algebra is the branch of mathematics concerned with studying linear sets of equations and their properties, and is very useful for describing and analysing transformations in space.

4 points

{3} Describe in words what each of the following transformation matrices do, when left-multiplied with the position vector $\mathbf{p}$ of a point:

$$\mathbf{T}_i\mathbf{p} = \mathbf{T}_i \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \mathbf{p}'.$$

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{T}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

4 points

{4} It is convenient to implement all affine transformations in 3D space using 4D homogeneous coordinates (as above). Describe how you would implement the transformation between an object's local coordinate frame (object space) and the world coordinate frame (world space) *without* using homogeneous coordinates. Also motivate why it is convenient to use homogeneous coordinates.

4 points

{5} 3D translation is not a linear operation, in the sense that there is no 3D matrix that can transform a 3D vector by translation only. Linear transformations are those that preserve lines and leave the origin unmoved. *Explain* how it is possible to implement 3D translation with a matrix multiplication using homogeneous coordinates.

Solution

{3} One point each:

- $\mathbf{T}_1$ does nothing (identity matrix).
- $\mathbf{T}_2$ translates 1 along z.
- $\mathbf{T}_3$ rotates $-90°$ counter-clockwise around z.
- $\mathbf{T}_4$ shears perpendicular to the z axis.

2 points

{4} Transforming an object from its local coordinate frame to the world frame is typically done by first rotating the object so that it has the desired orientation with respect to the world frame (change of basis), and then translate it.

With homogeneous coordinates, it is possible to store all transformations as matrices and append transformations just by multiplying the matrices together. Without homogeneous coordinates, translations would need to be implemented with vector addition rather than matrix multiplication, for example. So instead of concatenating rotation and translation as $\mathbf{p}' = \mathbf{Mp} = \mathbf{TRp}$, one would do $\mathbf{p}' = \mathbf{Rp} + \mathbf{t}$ (with $\mathbf{T}, \mathbf{R}$ transformation matrices representing translation and rotation, and $\mathbf{t}$ a column vector).

2 points

Using homogeneous coordinates allows all common operations — translation, rotation, scaling, as well as perspective projection — to be implemented as matrix operations. This means that any sequence of several translations and rotations can still be represented using a single matrix. Another advantage is that, using homogeneous coordinates, infinity (in any direction) can be represented with real numbers $[\mathbf{x}, 0]$.

{5} Using homogeneous coordinates, we extend the 3D xyz-space to the 4D $xyzw$-space, but we are only interested in the points that lie in the 3D subspace where $w = 1$. We can then apply a shearing transformation perpendicular to the w axis. Applying this shear to any point whose w coordinate was 1 will result in a transformed point that is still in the $w = 1$ plane, but has been translated by $[x, y, z]$.[3]

Question 4

Exact hard shadows can be easily implemented with ray tracing. However, for real-time rasterised graphics, two other popular classes of methods are shadow volumes and shadow maps.

4 points

{3} Outline the steps of a basic shadow mapping algorithm.

4 points

{4} List in total four advantages of shadow maps over shadow volumes, or vice versa, for real-time shadow rendering.

4 points {5} Augment your shadow mapping algorithm so that the algorithm also takes into account z fighting and shadow map aliasing. (Any of the methods mentioned in the lecture is fine, even if it's not the perfect solution to the problem.)

Solution

{3} The answer should include

1) Render scene from light source's position.
2) Save the corresponding depth buffer.
3) Render scene from camera view.
4) Transform each pixel's coordinates to light space (x_s, y_s, z_s).
5) If z_s (distance from light source) is greater than the corresponding (x_s, y_s) value in the shadow map, then the pixel is in shadow.

{4} One point for each of the items below.

Shadow maps

1) can generate shadows for any rasterisable geometry,
2) no need to compute silhouette edges,
3) no need to generate additional geometry,
4) no need to care about whether camera is in shadow or not,
5) faster (just one extra rendering pass per [directional] light source),
6) faster (doesn't affect fill rate).

Shadow volumes

1) have no issue with biasing,
2) have no issue with aliasing,
3) work well with omnidirectional light sources too.

{5} Up to two points for adding a depth bias to the test in step 5) above.

Up to two points for describing one of percentage closest filtering, shadow map fitting, warping or partitioning.

[3] I was fishing for an answer relating 3D translation to 4D shearing. But I realise that it wasn't quite clear from the question, so a more "concrete" answer, demonstrating that translation can be implemented with a 4D matrix multiplication, gives points too.