Datorgrafik DT3025

Martin Magnusson
January 12, 2017

- No tools (books or calculators) are allowed.
- Martin will come by at around 3 o'clock to answer questions.
- There are three types of questions on this exam, targeted for the assessment criteria that correspond to grades 3, 4, and 5.
  - To pass with grade 3 (G), you need 12 out of 16 points (75%) from the {3} questions **or** 20 points in total (from all questions).
  - To pass with grade 4 (VG), you need 12 points from the {3} questions **and** 12 points from either the {4} questions or the {5} questions.
  - To pass with grade 5, you need 12 points from all the categories {3}, {4}, and {5}.
- You may answer in Swedish or English.

Question 1

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 $(x_\mathrm{s}, y_\mathrm{s}, z_\mathrm{s})$.
5) If z_s (distance from light source) is greater than the corresponding $(x_\mathrm{s}, y_\mathrm{s})$ 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. Simply increasing the texture size is not really an augmentation of the algorithm, and that is what was asked for in the question.

Question 2

4 points

{3} What do the following terms stand for, and what are they used for in computer graphics?

1) BSDF

2) BRDF

3) BTDF

4) BSSRDF

4 points

{4} A person is walking home at sunrise, after a night at Kårhuset. The white walls of Entréhuset are lit by the rising sun. Show how you can compute the colour at a point P on the wall as observed from this position, under the following conditions, using the rendering equation.

- The sun is far enough so that it can be modelled as a directional light source. The incoming light direction can be represented by the unit vector $(0, 0.44, -0.90)$, in the observer coordinate system. (That is, the light from the sun is travelling in this direction.)
- (The observer coordinate system has x to the right, y up, and z backwards, or out from the page.)
- There is no other light source, and the effects of indirect lighting can be disregarded.
- The walls of Entréhuset are perfectly Lambertian, and white.
- The unit vector $(0, 0.31, -0.95)$ represents the direction from the observer to P.
- The normal vector at P is $(0, 0, 1)$.
- The intensity of the incoming light is represented by the RGB vector $(1, 0.9, 0.5)$.

Then what are the RGB colour components at P?

2 points {5} a) Based on your computation of the colour at P in the previous question, show what the computed colour would be for an observer standing 5 metres closer to the wall would be.[1]

2 points b) The diffuse and specular colors in the Blinn–Phong model are specified by RGB triples, but the specular exponent is expressed as a single value. Suppose that the model was divided up by wavelength so that the specular exponent could vary from colour to colour. Suppose you have a material modelled with a red exponent of 3, and blue and green exponents of 5, and that both the diffuse and glossy colours are pure white.

Describe in words the appearance of the specular highlight from a white point light, as well as the appearance outside of the highlight.

Solution

{3} Scattering functions:

1) BSDF: bidirectional scattering distribution function, is used to model the reflective and transmissive material properties of a material (in other words, the material's appearance under different lighting and viewing conditions, assuming that everything that's interesting happens at the surface).
2) BRDF: bidirectional reflectance distribution function, used to model the material properties of purely opaque surfaces.
3) BTDF: bidirectional transmission distribution function, used to model the material properties of purely transmissive surfaces.
4) BSSRDF: bidirectional sub-surface scattering reflectance distribution function, used to model the material properties for objects where important light effects also happens *inside* the material (e.g., marble, soap, milk).

{4} To compute the colour, let's use the rendering equation. Since there is just one light source and no indirect lighting, and a fully opaque surface, we can simplify the rendering equation to

$$L(P, \boldsymbol{\omega}_\mathrm{o}) = L(P, -\boldsymbol{\omega}_\mathrm{i})\, f_\mathrm{r}(P, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})\, (\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n})$$

[1] That is, wading in the fountain, as you would on such a morning…

and compute L separately for the red, green, and blue colour components.

We need the following quantities:

- $-\boldsymbol{\omega}_\mathrm{i} = (0, -0.44, -0.90)$
- $-\boldsymbol{\omega}_\mathrm{o} = (0, 0.31, -0.95)$
- $\mathbf{n} = (0, 0, 1)$
- $f_\mathrm{r}(P, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) = 1/\pi$ for all $\boldsymbol{\omega}_\mathrm{i}$ and $\boldsymbol{\omega}_\mathrm{o}$, since the surface is Lambertian. Let's also assume that f_r is the same for red, green, and blue (since the walls are white).
- $L(P, -\boldsymbol{\omega}_\mathrm{i}) = 1$ (red), 0.9 (green), 0.5 (blue).

With these numbers, the final term (the dot product) is

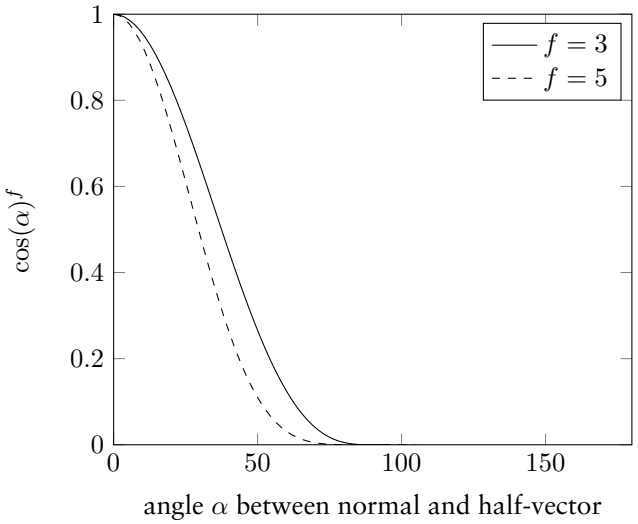
$$\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n} = 0 \cdot 0 + 0.44 \cdot 0 + 0.90 \cdot 1 = 0.90.$$

Then, the RGB colour at P is

- R: $1 \cdot 1/\pi \cdot 0.90 = 0.90/\pi$
- G: $0.9 \cdot 1/\pi \cdot 0.90 = 0.81/\pi$
- B: $0.5 \cdot 1/\pi \cdot 0.90 = 0.45/\pi$

{5} a) It would be the same. The colour of a Lambertian surface does not depend on the viewing angle, only on the angle to the light source.

b) The centre of the highlight would be white, but it would turn reddish towards the edges of the highlight, before turning white outside of the highlight. The reason is that the the blue and green curves are more peaked than the red one. See the plot below.



Question 3

4 points

{3} Outline, in pseudocode, the basic steps of a rasterisation algorithm.

{4} A rasteriser might do the following computations to check whether a pixel is inside a triangle or not:

1) Compute the edge equations $E_{1,2,3}$ for the triangle's three edges (given its three vertices)
$$E_i(x, y) = a_i x + b_i y + c_i.$$
(The vector (a_i, b_i) is along the direction of the normal vector of the edge.)
The parameters a, b, c can be computed as follows, given a pair of vertices (x_1, y_1) and (x_2, y_2):
$$\begin{aligned} a &= y_1 - y_2, \\ b &= x_2 - x_1, \\ c &= (x_1 - x_2)y_1 + (y_2 - y_1)x_1. \end{aligned}$$
Note that the vertices must be ordered *counter-clockwise* when computing E_i.
2) Evaluate each edge equation for the pixel position to see if the pixel is inside the halfspace given by each edge.

3 points

Given a triangle with vertices at $(0, 0)$, $(0, 10)$, and $(10, 0)$, go through the steps above to compute whether a pixel at $(4, 4)$ is in the triangle or not.

1 point

Describe what it is you compute if the vertices are not ordered counter-clockwise.

4 points

{5} Discuss ray casting and rasterisation from a performance standpoint. Which is more efficient, for which scenes?

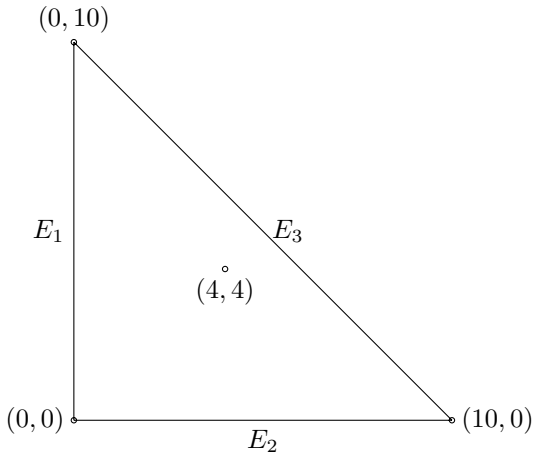
Solution

{3} Pseudocode for rasterisation:

```
for each triangle t do
    for each pixel p do
        if t covers p then
            Compute depth value z of t at p
            if z smaller than value of depth buffer at p then
                Replace depth buffer with z
                Compute pixel colour c
                Draw pixel colour to frame buffer
            end if
        end if
    end for
end for
```

The question was intended to be about a rasterising renderer for filled polygons (as in the follow-up questions {4} and {5}) but that may not have been clear from the question. Some people have also outlined a Bresenham-style line rasterisation algorithm. I have given 3 points for such answers.

{4} The triangle looks as in the figure below.



Computing a_i, b_i, c_i as in the instructions, we have

$$\begin{aligned} E_1(x, y) &= 10x + 0y + 0, \\ E_2(x, y) &= 0x + 10y + 0, \\ E_3(x, y) &= -10x - 10y + 100. \end{aligned}$$

Solving for $(4, 4)$, we have

$$\begin{aligned} E_1(4, 4) &= 40, \\ E_2(4, 4) &= 40, \\ E_3(4, 4) &= 20. \end{aligned}$$

Since all edge equations are positive, the pixel is inside the triangle.

If the edges are ordered clockwise instead, the condition that's computed is whether the point is on the "outside" of each edge's halfspace, which is *not possible*. This is not the same as a test to check if the point is outside the triangle.

{5} Rasterisation needs one triangle in memory at a time, plus frame and depth buffer (which take around 25 MiB). On the other hand, ray casting needs the entire scene in memory (which might include millions of triangles). Each *vertex* requires 3×4 bytes for position, and 12 more for normals; and then we have textures and face indices etc. However, with parametric geometry, memory requirements plummet. Also, parametric geometry is hard to impossible to rasterise!

The z-buffer used for rasterisation has worst-case $O(n)$ performance, while the ray-object intersection test used in ray casting is $O(\log n)$ if using a proper data structure (octree or kd-tree). So if there are lots (lots!) of overlapping polygons in the same pixel, rasterisation might become slower.

Methods used for casting shadows and rendering reflections can also affect the performance of rasterisation vs ray tracing; however, plain ray casting does not render shadows and reflections the way that ray tracing does.

Question 4

And now for some questions about algebra for computer graphics.

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

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

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

1 point b) Which of the following is a rotation-only matrix?

$$\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{R}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

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, but for translation, it is necessary to be able to move also points at the origin.

Why is it then that a matrix–vector multiplication using homogeneous coordinates can be used to implement 3D translation?

Solution

{3} One point each:

a)
- $\mathbf{T}_1$ scales along the z axis.
- $\mathbf{T}_2$ translates 1 along y.
- $\mathbf{T}_3$ rotates $-90°$ counter-clockwise around z.

Half a point if you only specify for example "scale" or "rotate" without saying which axis.

b) $\mathbf{R}_2$ is a rotation-only matrix — with zero angle! $\mathbf{R}_1$, on the other hand, both rotates and scales.

To get a point here, it is important that you recognise that $\mathbf{R}_1$ cannot be rotation only.

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 along 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 $[x, y, z]$.