Engineering Tripos Part IIA THIRD YEAR

Module 3G4: Medical Imaging & 3D Computer Graphics

Solutions to 2012 Tripos Paper

1. **Ultrasound imaging**

(a) (i) For the interface between water and fat

$$R_{wf} = \left(\frac{1.5 - 1.38}{1.5 + 1.38}\right)^2 = 1.736 \times 10^{-3}$$

(ii) For the interface between liver and fat

$$R_{lf} = \left(\frac{1.69 - 1.38}{1.69 + 1.38}\right)^2 = 10.196 \times 10^{-3}$$

[30%]

(b) (i) For path A. The attenuation in 20 cm of water at 7 MHz will reduce the intensity by a factor k where $-0.0022 \times 7 \times 20 = 10 \log_{10}(k)$ which gives us $k = 0.9315$. The reflection by the fat-water interface reduces the intensity by a factor of 1.736×10^{-3}. Travelling back through 20 cm of water reduces the intensity by 0.9315 again. This means that the overall proportion of the initial intensity that reaches the probe having been back-scattered by the water-fat interface is 1.5×10^{-3}.

(ii) For path B. This time we need the ultrasound intensity transmission coefficient between water and liver

$$T_{wl} = \frac{4 \times 1.5 \times 1.69}{(1.5 + 1.69)^2} = 0.996452$$

The attenuation in 10 cm of liver at 7 MHz will reduce the intensity by a factor k where $-0.7 \times 7 \times 10 = 10 \log_{10}(k)$ which gives us $k = 1.2589 \times 10^{-5}$.

So we have the following attenuation factors along path B.

attenuation by 10 cm of water at 7 MHz	0.965161
transmission through water-liver interface	0.996452
attenuation through 10 cm of liver at 7 MHz	1.2589×10^{-5}
reflection at liver–fat interface	10.196×10^{-3}
back through the liver	1.2589×10^{-5}
back through water-liver interface	0.996452
back through the 10 cm of water	0.965161

Multiplying all the numbers in the right-hand column together gives the total attenuation along path B as 1.495×10^{-12} [40%]

(c) (i) The sound will travel slower than assumed by the calibration of the ultrasound machine by a factor of $1540/1498 = 1.028$ therefore the distance to the water-fat interface

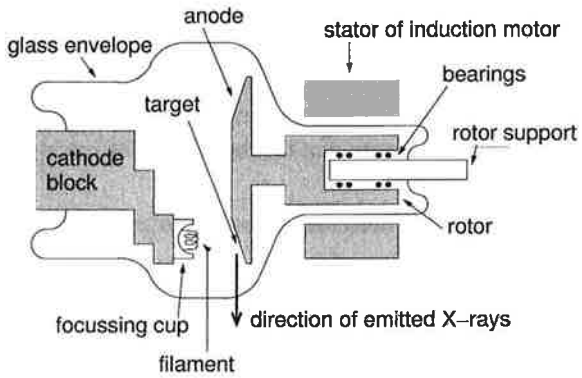
will be displayed too large by this factor by the ultrasound machine. The perceived depth is thus 20.56 cm.

(ii) For path B, the depth of the 10 cm in the water will be displayed as too big by the same factor as in the previous part, giving 10.28cm. However, as ultrasound travels faster than 1540 ms^{-1} in liver, this will appear too small by a factor of $1540/1604 = 0.96$. Hence this will appear to be 9.6 cm. The total depth to the liver-fat interface will thus be perceived as 19.88 cm. [30%]

Assessors' remarks: This question tested the candidates' understanding of attenuation and sound speed in ultrasonic imaging. It was relatively straightforward, but not easier that the norm in previous years. It was very popular and most candidates answered it extremely well. Many candidates got the correct numerical answer to the involved attenuation calculation in (b)(ii). In the sound speed calculations (c), there was an even higher proportion of correct answers.

2. **X-ray computed tomography**

(a)



X-rays are generated by bombarding a tungsten target (anode) with electrons. The cathode is heated by a high current (typically 10V, 10A), is incandescent at 2200° and emits electrons by thermionic emission. Electrons are repelled by the negative cathode and attracted by the anode some 50–125 KV more positive. The electrons accelerate in a vacuum and hit the cathode at half the speed of light. The tube current is controlled by the filament heating voltage. Less that 1% of the electron energy goes into the X-rays. The rest heats the tungsten anode which has a high melting point and good thermal conductivity to conduct the heat away efficiently. The anode also rotates to give each point of electron impact time to cool. [30%]

(b) Three properties of X-rays that can be used to detect them: they can ionise matter, they make some materials fluoresce, and they affect photographic emulsion. [15%]

(c) Derivation of *filtered backprojection* from the *projection theorem*.

We need to find a way of inverting the Radon transform:

$$\mu(x,y) = \mathcal{R}^{-1}\left[p(s,\phi)\right]$$

$\mu(x,y)$	The X-ray attenuation values of the object that we want to reconstruct.
$\mathcal{R}^{-1}$	The inverse Radon transform
$p(s,\phi)$	Projection values (from the CT machine).

The projection theorem states that the 1D Fourier transform ($\mathcal{F}_{1(\omega)}[\,]$) of the projection data $p_\phi(s)$ at a given projection angle ϕ is the same as the radial data passing through the origin at a given angle ϕ in the 2D Fourier transform ($\mathcal{F}[\,]$) of the attenuation data $\mu(x,y)$.

$$\mathcal{F}_{1(\omega)}\left[p_\phi\right] = \mathcal{F}\left[\mu\right]$$

Take the inverse 2D Fourier transform of both sides:

$$\mu(x,y) \;=\; \int\!\!\int_{-\infty}^{+\infty} \mathcal{F}_{1(\omega)}\left[p_\phi\right] e^{i(\omega_x x+\omega_y y)} d\omega_x\, d\omega_y$$

Next , we use a Jacobian to rewrite the integral in terms of polar (ω,ϕ) instead of rectangular (ω_x,ω_y) coordinates in the spatial frequency domain:

$$\begin{aligned} \omega_x &= \omega\cos\phi\,,\quad \omega_y = \omega\sin\phi \\ J &= \left|\begin{array}{cc} \partial\omega_x/\partial\omega & \partial\omega_x/\partial\phi \\ \partial\omega_y/\partial\omega & \partial\omega_y/\partial\phi \end{array}\right| = \left|\begin{array}{cc} \cos\phi & -\omega\sin\phi \\ \sin\phi & \omega\cos\phi \end{array}\right| = \omega \end{aligned}$$

Hence:

$$\mu(x,y) = \int_0^{\pi}\left\{\int_{-\infty}^{+\infty} \mathcal{F}_{1(\omega)}\left[p_\phi\right]|\omega| e^{i\omega(x\cos\phi+y\sin\phi)} d\omega\right\} d\phi$$

The inner integral is a one-dimensional inverse Fourier transform in the s direction at constant ϕ:

$$\begin{aligned} \mu(x,y) &= \int_0^{\pi}\left\{\int_{-\infty}^{+\infty} \mathcal{F}_{1(\omega)}\left[p_\phi\right]|\omega| e^{i\omega s} d\omega\right\} d\phi \\ &= \int_0^{\pi}\left\{\mathcal{F}^{-1}_{1(s)}\left[\,\mathcal{F}_{1(\omega)}\left[p_\phi\right]|\omega|\,\right]\right\} d\phi \end{aligned}$$

Multiplication in the frequency domain is equivalent to convolution in the spatial domain, so

$$\mathcal{F}_{1(\omega)}\left[p_\phi(s)\right] \times |\omega| \leftrightarrow p_\phi(s) * q(s)$$

where $q(s)$ is the inverse Fourier transform of $|\omega|$:

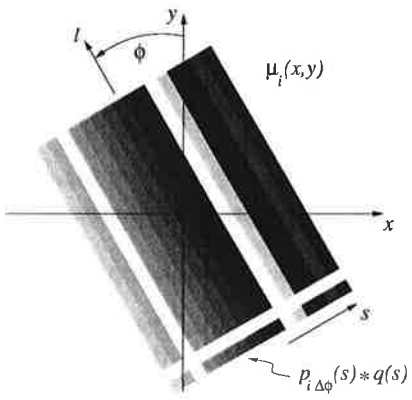
$$q(s) = \mathcal{F}^{-1}_{1(s)}\left[|\omega|\right] = \int_{-\infty}^{+\infty} |\omega| e^{i\omega s} d\omega$$

Hence:

$$\mu(x,y) = \int_0^{\pi} p_\phi(s) * q(s)\ d\phi$$

The above equation is the essence of the filtered backprojection algorithm, which is best understood by considering a practical implementation where the integral is replaced by a finite summation:

$$\mu(x,y) = \sum_{i=0}^{n-1} \{p_{i\Delta\phi}(s) * q(s)\}$$



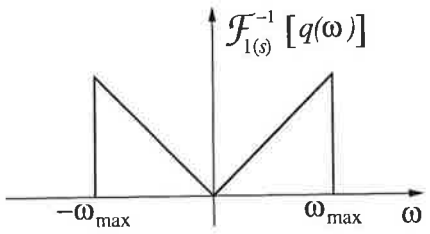
1 Take n projections $p_\phi(s)$ of the object.

2 Convolve each projection with $q(s)$.

3 Backproject each filtered projection in the l direction.

4 Accumulate the backprojections. [35%]

(d) The filter kernel $q(s)$ is divergent in frequency and therefore unattainable in practice. However, the X-ray beam has a finite width t, which means that the useful Fourier content is limited to frequencies below $\omega_{\max} = 2\pi/t$. The filter can therefore be cut off at $\omega = \omega_{\max}$. The resulting filter, shown below, is called the Ram-Lak filter after its inventors Ramachandran and Lakshiminarayanan.



Often, frequencies just below ω_{max} are unreliable because of aliasing and noise, so a smoothing window (e.g. Hamming) is applied to the Ram-Lak filter. [20%]

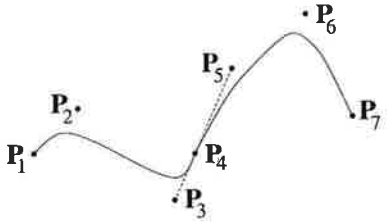
Assessors' remarks: This question tested the candidates' knowledge of X-ray computed tomography. It was a straightforward question in which success was to a large extent dependent on an ability to reproduce book work from the lecture notes. It was unpopular and those candidates who did attempt it generally produced poor solutions.

3. **Parametric cubic curves and continuity**

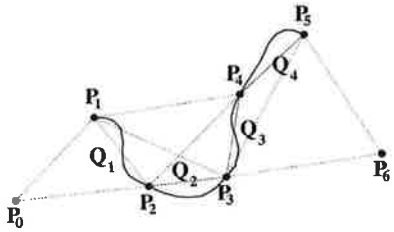
(a) A Bézier curve is defined by four control points, two of which define the ends of the curve and the other two define the end gradients. It is fairly easy to create a single curve segment, but only provides continuity between segments if neighbouring points are carefully aligned. It can easily be sub-divided, which makes it easy to refine a pre-existing curve. It is also useful for display, since the curve is within the convex hull of the control points, so if a polygon formed from the control points is outside the visible area, the curve does not need to be drawn. It can also be efficiently displayed by iterative sub-division and then joining up the resulting control points.

A Catmull-Rom curve will always pass through the control points, and a multiple segment curve inherently has first order continuity. Hence it is easy to define a line which is fairly smooth and passes through defined points. However, it does not have the same convex hull property as the Bézier nor can easily be sub-divided, so conversion is probably necessary before refinement or display. However, the multi-segment nature means moving one control point still results in the same continuity.

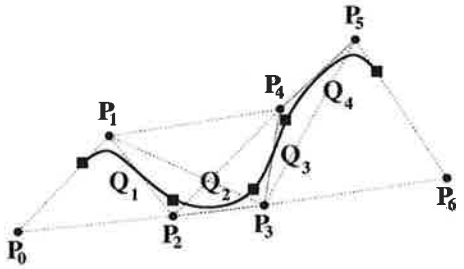
A B-spline curve does not pass through control points, but it does pass quite close to them and exhibits second order continuity. So it is a good choice when defining a smooth curve if it is not critical exactly where the curve is located. It does have the convex hull property, so can be displayed according to whether the control point polygon is visible or not. Again, it is probably easier to convert to a Bézier for curve refinement, although like the Catmull-Rom the control points can easily be moved without changing the overall continuity. [30%]



(b) Continuity between curve segments of a Bézier curve is only achieved by careful matching of neighbouring control points. Sharing (for instance) $\mathbf{p}_4$ gives zeroth order parametric and geometric continuity, Having the neighbouring control points ($\mathbf{p}_3$ and $\mathbf{p}_5$) in a straight line through $\mathbf{p}_4$ gives first order continuity, which is parametric if $\mathbf{p}_4$ is in the centre of this line.



The Catmull-Rom curve has inherent first order geometric continuity, due to the sharing of three control points with neighbouring curve segments. This is unless there are coincident control points, in which case geometric continuity can be reduced to zeroth order.



The B-spline has inherent second order geometric continuity, again due to sharing of control points between segments. As with the Catmull-Rom, geometric continuity can be reduced if control points are coincident. [15%]

(c) (i) The curve is defined as a Catmull-Rom so we already know it has at least first order continuity. For second order continuity we need the second derivative of the curve:

$$\frac{1}{2}\begin{bmatrix} 6t & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{p}_4 \end{bmatrix}$$

For the end of the first curve segment at $\mathbf{p}_3$, we use $t = 1$ and the points $\mathbf{p}_1 \ldots \mathbf{p}_4$ which gives the second derivative as:

$$-\mathbf{p}_1 + 4\mathbf{p}_2 - 5\mathbf{p}_3 + 2\mathbf{p}_4$$

For the start of the second curve segment at $\mathbf{p}_3$, we use $t = 0$ and the points $\mathbf{p}_2 \ldots \mathbf{p}_5$ which gives the second derivative as:

$$2\mathbf{p}_2 - 5\mathbf{p}_3 + 4\mathbf{p}_4 - \mathbf{p}_5$$

Equating these two gives the condition as:

$$\mathbf{p}_1 - 2\mathbf{p}_2 + 2\mathbf{p}_4 - \mathbf{p}_5 = 0 \qquad [20\%]$$

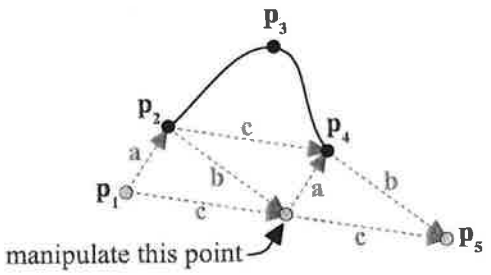
(ii) Defining these vectors gives:

$$\begin{aligned} \mathbf{a} &= \mathbf{p}_2 - \mathbf{p}_1 \\ \mathbf{b} &= \mathbf{p}_5 - \mathbf{p}_4 \\ \mathbf{c} &= \mathbf{p}_4 - \mathbf{p}_2 \end{aligned}$$

Substituting these into the condition from (i) gives:

$$\mathbf{c} = \mathbf{a} + \mathbf{b}$$

Hence we need to manipulate $\mathbf{p}_1$ and $\mathbf{p}_5$ whilst keeping $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ forming a triangle between them.



The diagram above shows a potential way to do this by just moving a single point. This will ensure that the curve still interpolates the three points $\mathbf{p}_2 \ldots \mathbf{p}_4$ with continuity in the second derivative. [20%]

(iii) For a three-segment spline, there would be the same condition on $\mathbf{p}_1$ and $\mathbf{p}_5$: however $\mathbf{p}_5$ would then be the last point to be interpolated, and hence fixed. This implies that $\mathbf{p}_1$ would also be fixed. By a similar argument, the location of $\mathbf{p}_2$ also fixes the location of the

new point $\mathbf{p}_6$. Hence it *is* possible to interpolate four points with continuity in the second derivative, however, there is no longer any flexibility in the location of the extreme control points $\mathbf{p}_1$ and $\mathbf{p}_6$, and as a result only one possible curve. [15%]

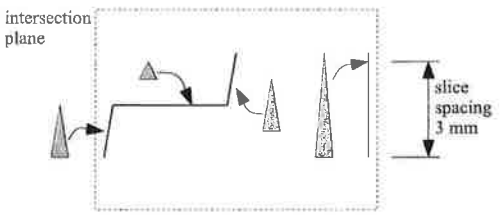
Assessors' remarks: This question concerned the differences between Bezier, B-spline and Catmull-Rom curves, particularly regarding continuity. This was a fairly popular question with generally good answers to the background material in (a) and (b). Answers to (c) were much more variable. There were some perfect solutions, but many students lost marks due to mistakes in arithmetic as opposed to technique. There were also several attempts to preserve the continuity of the control points rather than the actual curve.

4. **Marching cubes, distance transforms and linear interpolation**

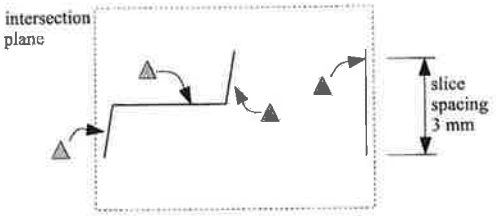
(a) The Marching Cubes technique works by considering a single cube of the data at a time, with one data sample at each cube vertex, giving eight data points in all. A triangulated surface is constructed for each of these cubes, which completely separates cube vertices with data above the iso-surface threshold from cube vertices with data below the iso-surface threshold. This operation is repeated by 'marching' the cube over the entire data volume, and generating these surface patches for every cube in the volume. The total set of surface patches is the triangulated iso-surface for the whole volume.

At each cube, the surface is designed by noting which cube vertices have data above or below the threshold. This gives 256 possible cases, which can be reduced to 15 by considering symmetry. A look-up table is constructed for these 15 cases which stores the number and arrangement of triangles to use for this case, and which cube *edges* the triangle vertices should be placed on.

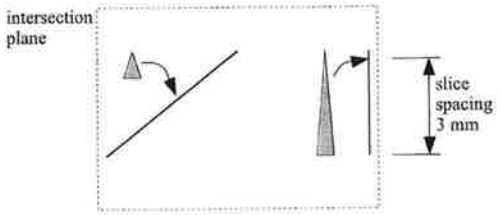
The final step is to work out exactly where on the cube edge each vertex should be placed. This is done by linear interpolation of the data values at each end of this edge: by definition these will always be either side of the iso-surface threshold, since otherwise there would not be a triangle vertex at this edge. For instance, if the data values are 1 and 5, and the threshold was at 3, the vertex would be placed exactly half way along the edge. [25%]



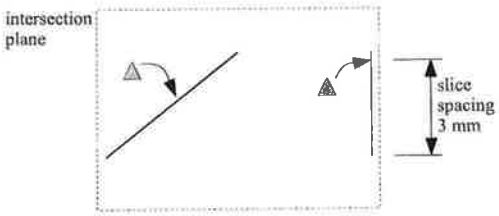
(b) (i) The diagram above shows the intersection of the surface as a solid line. Marching Cubes involves linear interpolation of the data values: on the left this will give the threshold value of 50 half way between the lower and upper layers. Triangles tend to be similar in size to the 'cubes', which in this case are 3 mm high but only 0.5 mm wide. So triangles on the vertical surfaces will be very long with poor aspect ratio, whereas triangles on the horizontal surfaces will tend to be smaller and more regular.



(ii) As the diagram above shows, linear interpolation makes no difference to the actual surface location: Marching Cubes was already using linear interpolation in (i). However, the addition of some intermediate surfaces means that the voxels are now genuinely cubes, and hence all the triangles will now have similar sizes.



(iii) Distance transformation replaces data values with the distance to the edges, and this will result in the much more plausible surface intersection shown above. However, we are still using very stretched 'cubes' for Marching Cubes, so the triangle size will vary. On the vertical surface there are still very long triangles, whereas the other surface has slightly extended triangles (due to the slope).



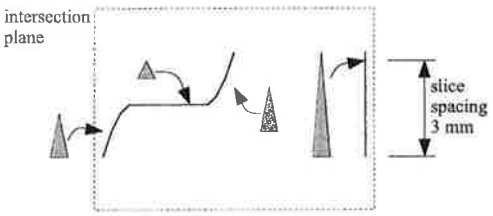
(iv) This scheme combines the benefit of correct surface location in (iii) with the creation of sensible size triangles as in (ii), this time across the entire surface. [40%]

(c) Surfaces created by Marching Cubes tend to contain a lot of very small triangles, all of a similar size to the dimensions of each cube used in the technique. In areas where the iso-surface is quite flat there is no need to have so many triangles, hence the surface mesh is often processed to reduce the number of triangles in these regions. Such techniques seek to iteratively remove vertices (and then re-triangulate the resulting holes) where the

removal has little impact on surface accuracy. This results in much bigger triangles over flat regions, but smaller triangles over more curved regions.

Marching Cubes also generates some triangles with very poor aspect ratios, which can cause problems when visualising the iso-surface. These can also be removed by post-processing with techniques as outlined above. [15%]

(d) The answer to (b) (iii) would actually be identical: blurring extends the sharp discontinuity in both directions, but the *centre* of the discontinuity remains in the same location. Since we are thresholding at 50, which is at the exact centre of the edge, the threshold location stays in the same place. After thresholding we then throw away all the other blurred data and replace it with the *same* distance transform as before.



On the other hand, the answer to (b) (i) is affected by image blurring, creating the slightly softer surface intersection shown in the diagram above. The actual form of this intersection will depend on the extent of the blur compared to the size of the surface feature. [20%]

Assessors' remarks: This question concerned the extraction of isosurfaces from volume data. The background material in (a) and (c) was answered very well. There were some excellent answers to (b), though some confusion about the difference between using a distance transform and simple thresholding. Several students spotted in (d) that blurring does not have any significant effect if it is followed by thresholding.

5. **The Phong model and surface rendering**

(a) I_λ is the intensity of the reflected light of colour λ, where $\lambda \in \{r, g, b\}$ for red, green and blue.

I_λ depends on several terms. First, there is the ambient reflection term, $c_\lambda I_a k_a$, which models indirect illumination of the surface. c_λ, where $0 \leq c_\lambda \leq 1$, specifies the colour of the surface. I_a is the intensity of the general background illumination, and k_a is the surface's ambient reflection coefficient.

The next two terms in the model are calculated for a point light with intensity I_p. First there is the diffuse reflection term, $c_\lambda k_d \mathbf{L}.\mathbf{N}$, which models even reflection of the light source in all directions. Diffuse reflection is greatest when the surface is pointing directly towards the light source, and tails away to zero when the surface is side-on to the light source. $\mathbf{L}$ is the unit vector from the surface point towards the light source, $\mathbf{N}$ is the unit surface normal

and k_d is the surface's diffuse reflection coefficient (small for dark surfaces, high for bright surfaces).

Finally, there is the specular reflection term, $k_s\,(\mathbf{R}.\mathbf{V})^n$, which models directional reflection of the light source along the unit mirror vector **R**. **V** is the unit vector from the surface point towards the viewer. The viewer only perceives the specular highlight (or glint) when looking along the mirror direction, or at least close to it. k_s is the surface's specular reflection coefficient (small for matte surfaces, high for shiny surfaces), and n is the specular exponent that determines the tightness of the glint. n is high for a tight highlight (e.g. a perfect mirror) and small for a more blurred highlight (e.g. aluminium).

The model can be extended to incorporate multiple light sources i, depth cueing and shadows as follows:

$$I_\lambda = c_\lambda I_a k_a + \sum_i S_i f_{\text{att}} I_{pi}(c_\lambda k_d \mathbf{L}_i.\mathbf{N} + k_s\,(\mathbf{R}_i.\mathbf{V})^n)\ .$$

The diffuse and specular terms are attenuated by a shadow factor S_i, where $0 \leq S_i \leq 1$. S_i is the fraction of the pixel shaded from the light source, often calculated using a shadow Z-buffer algorithm. There is also a depth attenuation factor f_{att}, usually of the form

$$f_{\text{att}} = \min\left(\frac{1}{a_1 + a_2 d + a_3 d^2}\ ,\ 1\right)$$

where a_1, a_2 and a_3 are constants, and d is the distance from the light source to the surface point. The depth cueing ensures that surfaces with the same orientation, but at different distances from the viewer, are not assigned the same intensity. [40%]

(b) Calculating **R** involves considerable computational expense. The advantage of the **N**.**H** alternative (which produces very similar specular highlights) becomes apparent when the light source and viewer are both at infinity. Under these conditions, **L**, **V** and **H** are constant across the scene. So calculating the specular component requires only a single dot product **N**.**H** for each vertex. [10%]

(c) (i) The rendering on the left uses flat shading and, conceivably, orthographic projection. The alternative would be perspective projection with almost parallel projectors, consistent with a distant target and a long focal length. The rendering in the middle uses the same projection and Gouraud shading. The rendering on the right uses Gouraud shading and perspective projection: there is significant perspective compression from the front to the back of the 'O'. [20%]

(ii) All three renderings suffer from:

Lack of self-shadowing. Given the position of the light source, the interior of the 'O' should be entirely in shadow. This could be fixed using an extended Phong model, as in (a), in conjunction with a shadow z-buffer algorithm.

Saturation. The specular highlights are saturated white and bleed into the background. This could be fixed by reducing the illumination intensity to ensure that no Phong calculations saturate.

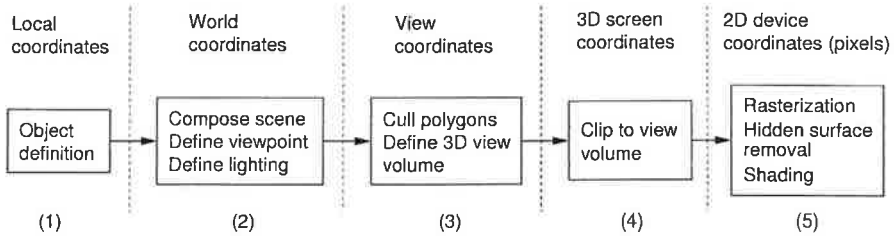
Uniformly shaded side surfaces. The side surfaces are illuminated entirely by the Phong ambient term and therefore appear unconvincingly flat. This could be fixed by adding another light source so that every polygon reflects some diffuse illumination.

Jagged edges. Close inspection of the renderings shows that the jagged edges have nothing to do with the underlying polygon structure of the 'O', so increasing the number of polygons will not help. What we are seeing are individual pixels in the framebuffer. The simplest way to anti-alias the renderings is through *supersampling* or *postfiltering*. This involves rendering the image into an intermediate framebuffer at several times the display resolution. The supersampled image is then low-pass filtered at the Nyquist limit of the final display framebuffer, which is filled by subsampling the intermediate framebuffer. [30%]

Assessors' remarks: This question assessed the candidates' understanding of the Phong model and its use in rendering algorithms. Part (a) was book work and was very well answered by almost all the candidates. In (b), most candidates stated that using the halfway vector would be quicker, but very few explained that this would only work when the light source and viewer were at infinity. In (c)(i), most were able to explain why the renderings differed but nobody identified all the artefacts in (c)(ii). It was pleasing, however, to see several candidates correctly attributing the jagged edges to rasterization as opposed to polygons, and suggesting appropriate anti-aliasing remedies.

6. **Clipping, hidden surface removal and surface rendering**

(a)

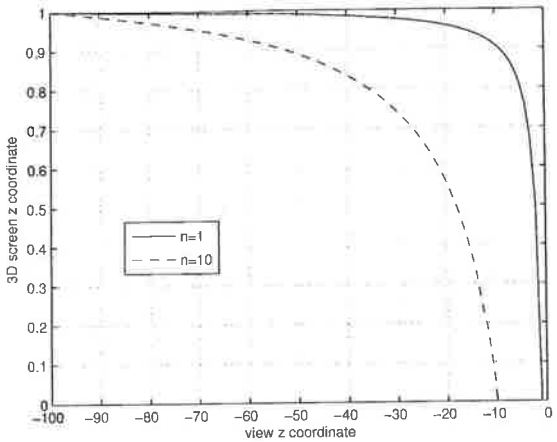


[25%]

(b) (i) It is important that the mapping takes this form, since it means that the transformation between view and 3D screen coordinates is a projective one. This, in turn, guarantees that lines map to lines and planes map to planes, which is essential if we are to use any sort of linear interpolation at later stages of the rendering pipeline. 4×4 matrix operations are ubiquitous in computer graphics and widely supported in hardware. [10%]

(ii) Clipping to the view volume can be performed in homogeneous coordinates, so the expensive divide-by-w operation is necessary only for those vertices that survive the clipping process. Further vertices may have been discarded even earlier by back-face culling. [10%]

(iii)



Candidates are expected to produce just a single, representative curve. Here we show curves for $f = 100$ and $n \in \{1, 10\}$, to illustrate answers to (c) and (d). [10%]

(c) With reference to the curves in (b), depth discrimination becomes more problematic near the far clipping plane, where a larger range of z_v values map to similar z_s values. So we should look at the values $z_s = 1$ and $z_s = 1 - 2^{-16} + \epsilon$ (where ϵ is vanishingly small), since these will both be stored as 2^{16} in the hardware z-buffer. $z_s = 1$ clearly maps to $z_v = -100$, and

$$z_s = 1 - 2^{-16} = \frac{f(1 + n/z_v)}{f - n} = \frac{100(1 + 1/z_v)}{99} \Leftrightarrow z_v = 99.849$$

The maximum difference between the two z_v values is therefore $100 - 99.849 = 0.151$. [30%]

(d) We could increase the resolution of the z-buffer beyond 16 bits, but this would require hardware modifications. A far simpler fix would be to increase n, since it is the ratio of f/n that determines the amount of perspective compression. For example, increasing n from 1 to 10 (see graph in (b)) improves matters considerably: the answer to (c) becomes 0.0137. The only side effect would be nearby objects disappearing behind the camera (or, to be precise, behind the near clipping plane) a little sooner. [15%]

Assessors' remarks: This question assessed the candidates' understanding of the various coordinate systems used in the surface rendering pipeline, particularly 3D screen coordinates and their role in clipping and hidden surface removal. Part (a) was book work and was very well answered by almost all the candidates. In (b), most showed a solid understanding of clipping though several thought that vertices might be clipped because they were obscured by other polygons, not because they were outside the view volume. Many candidates performed the calculations correctly in (c) and got full marks, most of the others at least appreciated that the issues would be at the far clipping plane. In (d), most

candidates made sensible suggestions involving the z-buffer precision and/or the clipping plane positions, though most missed the pertinent point that it is the ratio f/n that matters most.

Andrew Gee, Richard Prager & Graham Treece
May 2012

Part IIA 2012

Module 3G4: Medical Imaging & 3D Computer Graphics

Numerical Answers

1. (a) (i) 1.763×10^{-3}, (ii) 10.196×10^{-3}

   (b) (i) 1.5×10^{-3}, (ii) 1.495×10^{-12}

   (c) (i) 20.56 cm, (ii) 19.88 cm

6. (c) 0.151