Engineering Tripos Part IIA THIRD YEAR

Module 3G4: Medical Imaging & 3D Computer Graphics

Solutions to 2026 Tripos Paper

1. **Ultrasound and MRI**

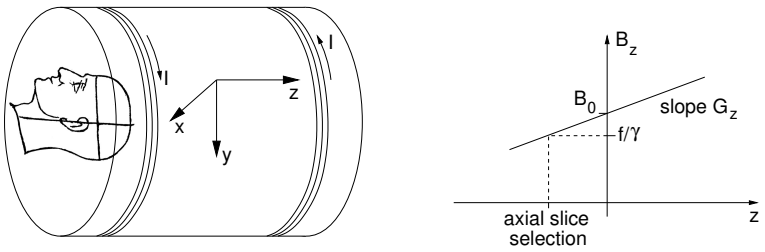
(a) Since bone has a very different acoustic impedance to soft tissue, and a very high attenuation coefficient, very little of the transmitted ultrasound wave will be able to penetrate the skull and reach the brain. For this reason, MRI is normally preferred. The exceptions are neonatal brain imaging (scanning through the fontanelle) and intra-operative imaging (following craniotomy). [10%]

(b) (i) This is the *spin-echo* imaging sequence, which involves two excitations. First a 90° radio-frequency pulse is used to initiate free induction decay. Once the spin vectors have dephased due to deterministic T_2^* relaxation, a 180° pulse is applied to change their directions. The faster vectors are now at the back of the pack and the slow ones are at the front. The vectors will therefore come back into phase and produce an "echo" signal that is not dominated by T_2^* effects. [20%]

(ii) An MRI scanner incorporates three gradient coils that can apply an arbitrary linear variation to the field in each of the principal scanning directions:

$$\frac{\partial B_z}{\partial z} = G_z \ , \quad \frac{\partial B_z}{\partial x} = G_x \ , \quad \frac{\partial B_z}{\partial y} = G_y$$

For example, the z-gradient might look like this:



In the sequence in Fig. 1 of the question, G_z is applied at the same time as the RF pulse. This restricts imaging to a particular **axial** slice, depending on the frequency f of the pulse, since only protons exposed to a field f/γ are excited. [15%]

(iii) Apart from slice selection, we can use the gradient fields for phase and frequency encoding, allowing us to infer where in a slice a particular NMR signal comes from. Continuing the example in Fig. 1, by applying G_y after the RF pulse, the precessions will go at different speeds according to their y-coordinates. We then switch off the y-gradient, so the precessions return to their original speeds, but their phases now encode the y-coordinate. [10%]

(iv) We then apply G_x at the same time as we detect the NMR signal. The x-gradient causes protons to precess at different speeds according to their x-coordinates. So, when we detect the signal, frequency encodes x. However, G_x will affect not only frequency but also phase during detection. This unwanted de-phasing can be undone by applying a compensating G_x gradient before the 180° pulse. [20%]

(v) $I(x, y)$ is obviously dependent on proton density in a time-independent manner. Furthermore, it will decay exponentially at a rate governed by T_2 because of spin-spin relaxation. $I(x, y)$ does not depend on T_2^* thanks to the spin-echo sequence. The T_1 dependence comes from the pulse repetition period TR. We need to repeat the sequence many times, with different phase gradients, to form a complete (x, y) image. If TR is short, then little T_1 relaxation occurs before the next 90° pulse, the z-field does not fully recover, and we effectively start over with a lower value of B_0. Hence the $(1 - e^{-\mathrm{TR}/T_1(x,y)})$ term.

By tuning TR and TE, we can emphasize different terms in the equation for $I(x, y)$:

PD-weighted: Long TR, short TE.

$\mathbf{T_1}$-weighted: Short TE, TR $\approx T_1$

$\mathbf{T_2}$-weighted: Long TR, TE $\approx T_2$ [25%]

Assessor's remarks: This question tested candidates' understanding of MRI fundamentals in general and the spin-echo imaging sequence in particular. It was mostly book work but also required some comparative analysis with ultrasound in (a) and engagement with the flipped classroom sessions and/or Moodle quizzes in (b)(iv). The better responses were those that addressed the questions directly and succinctly. For example, all that was required in (a) were a couple of sentences mentioning bone, attenuation and acoustic impedance. There were many responses that were simply lecture note dumps, hoping to stumble upon a salient point. Such responses received little credit.

2. **Multi-segment parametric cubic spline loops**

(a) Multi-segment parametric cubic splines are formed from multiple individual parametric curves in t, each limited to the range $0 < t < 1$. A curve ending at $t = 1$ is joined to the next at $t = 0$, so the continuity at each of these joins defines how smooth the overall multi-segment curve is.

Parametric continuity is the continuity of the parameter t. C^0 continuity implies that the curves are joined, C^n continuity implies that the nth derivative of t is equal at the join point.

Geometric continuity is the continuity that you can see, or equivalently the continuity of the coordinates (e.g. x and y). G^0 again means the curves are joined, G^1 continuity means the directions of the tangent vectors are equal at the join point.

We need different degrees of continuity for different situations. If we only want the curve to look reasonably smooth, G^1 is sufficient. If this is a motion path and we want smooth motion, then C^2 continuity is required. [20%]

(b) B-splines have inherent C^2 continuity and Catmull-Rom splines have inherent C^1 continuity. Normally they both have the same geometric as parametric continuity. However, in both cases, if some control points are coincident, the *geometric* continuity can be reduced, while the *parametric* continuity stays the same. [10%]

(c) (i) In order for the curve to interpolate the points, we need to return the second control point in the geometry matrix at $t = 0$ and the third control point for $t = 1$. For segment $\mathbf{S}_0$:

$$\mathbf{S}_0(t) = \frac{1}{4}\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{p}_3 \\ \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix}$$

For $t = 0$:

$$\frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

which will return $\mathbf{p}_0$. For $t = 1$, either use $\mathbf{S}_0^r(0)$ with the result above or, directly:

$$\frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$$

which will return $\mathbf{p}_1$, so this is an interpolating curve. [15%]

(ii) To calculate the parametric continuity (geometric continuity is assumed the same for non-coincident points), we need **TM** for $t = 0$ and $t = 1$ and for the first and second derivatives of **T**.

For the first derivative:

$$\mathbf{T}' = \frac{1}{4}\begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}$$

If we look at continuity around the point $\mathbf{p}_1$, for $t = 0$:

$$\frac{1}{4}\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -\frac{3}{4} & 0 & \frac{3}{4} & 0 \end{bmatrix} \quad (1)$$

which will return $\frac{3}{4}(\mathbf{p}_2 - \mathbf{p}_0)$ for $\mathbf{S}_1$. For $t = 1$:

$$\frac{1}{4}\begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{3}{4} & 0 & \frac{3}{4} \end{bmatrix} \quad (2)$$

which will return $\frac{3}{4}(\mathbf{p}_2 - \mathbf{p}_0)$ for $\mathbf{S}_0$, so C^1 continuity is achieved.

For the second derivative:

$$\mathbf{T}'' = \frac{1}{4}\begin{bmatrix} 6t & 2 & 0 & 0 \end{bmatrix}$$

If we again look at continuity around the point $\mathbf{p}_1$, for $t = 0$:

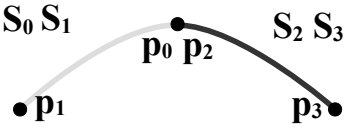
$$\frac{1}{4}\begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & -\frac{9}{2} & 3 & -\frac{3}{2} \end{bmatrix} \tag{3}$$

which will return $\frac{1}{2}(6\mathbf{p}_0 - 9\mathbf{p}_1 + 6\mathbf{p}_2 - 3\mathbf{p}_3)$ for $\mathbf{S}_1$. For $t = 1$:

$$\frac{1}{4}\begin{bmatrix} 6 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} -3 & 5 & -5 & 3 \\ 6 & -9 & 6 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -\frac{3}{2} & 3 & -\frac{9}{2} & 3 \end{bmatrix} \tag{4}$$

which will return $\frac{1}{2}(-3\mathbf{p}_3 + 6\mathbf{p}_0 - 9\mathbf{p}_1 + 6\mathbf{p}_2)$ for $\mathbf{S}_0$, so C^2 continuity is also achieved. [30%]

(iii) If $\mathbf{p}_0 = \mathbf{p}_2$, then two of the segments will be identical to the other two, and this will represent a curve (drawn twice) rather than a loop:



The easiest way to see this is to compare $\mathbf{G}_0$ with $\mathbf{G}_1$, with $\mathbf{p}_2 = \mathbf{p}_0$:

$$\mathbf{G}_0 = \begin{bmatrix} \mathbf{p}_3 \\ \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_3 \\ \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_0 \end{bmatrix}; \qquad \mathbf{G}_1 = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_0 \\ \mathbf{p}_3 \end{bmatrix} = \mathbf{G}_0^r$$

Hence $\mathbf{G}_1$ is just $\mathbf{G}_0$ reversed, and will generate a curve in exactly the same place.

The parametric continuity stays the same, but the geometric continuity is now only G^0. [15%]

(iv) For loops of more than four points, continuity would only be achieved up to C_1.

For two joining segments of a multi-segment curve, only three points are normally shared, so continuity is only guaranteed if the non-shared points are not involved at all in that derivative. Hence the zeros at the end of equation (1) and start of (2). Equations (3) and (4) do not have these zeros, and hence the non-shared points are involved: it only has C^2 continuity because the whole loop only has four points, and hence the normally not shared points ($\mathbf{p}_3$ in this example) are actually the same for this loop. [10%]

Assessor's remarks: This was a question mainly on continuity of multi-segment parametric cubic curves. There were many good answers, including quite a few noting that the

C^2 continuity was reliant on the loop having only four points. Most candidates could state the expected continuity for B-splines and C-R splines and demonstrate the required continuities for the new type of curve in (c)(i–ii). Marks tended to be lost with imprecise or inaccurate definitions of parametric and geometric continuity in (a), and incorrect sketches in (iii) due to the lack of any maths to support them.

3. **Interpolation of unstructured 2D data**

(a) In nearest neighbour interpolation, the value of the closest data points to the requested point is returned. In linear interpolation, the closest data points (which span the requested point) are weighted using a linear function, where the weights add up to one and are all between zero and one.

For unstructured data, a Delaunay triangulation imposes a geometrical framework on this data where each triangle connects three of the data points. Linear interpolation can then be used within each triangle to determine values from the data points at the corners of these triangles. This uses barycentric coordinates: values are interpolated along two triangle edges first, and then interpolated from these to the point.

For nearest neighbour interpolation, we need a Voronoi tessellation, which can be derived from the Delaunay triangulation by finding the mid-point of each triangle edge and drawing perpendicular lines from these points which connect with each other. Throughout each Voronoi cell we then return the data value from the point contained by that cell. [20%]

(b) (i) Note that the given data points form an equilateral triangle with side length 2, which is easily demonstrated since $\sqrt{3+1} = 2$. The given matrix equation with specific values inserted is hence:

$$\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

This leads to the equations:

$$\begin{aligned} 2(\lambda_2 + \lambda_3) &= 0 \\ 2(\lambda_1 + \lambda_3) &= 1 \\ 2(\lambda_1 + \lambda_2) &= 1 \end{aligned}$$

with solution $\lambda_1 = \frac{1}{2}$, $\lambda_2 = \lambda_3 = 0$.

Hence the RBF is:

$$s(\mathbf{p}) = \frac{1}{2}\phi(||\mathbf{p} - \mathbf{p}_1||) = \frac{||\mathbf{p}||}{2}$$ [30%]

(ii) This is not the same result as a linear interpolation of the Delaunary triangulation: which in this case is just a triangle with $\mathbf{p}_1$ to $\mathbf{p}_3$ at the vertices. This is easily demonstrated by considering the point $\mathbf{p} = (\sqrt{3}, 0)$.

For a linear interpolation within the triangle, this would return the value 1, since this point is along the side connecting $\mathbf{p}_2$ and $\mathbf{p}_3$ which both have value 1.

For the RBF:

$$s(\mathbf{p}) = \frac{||\mathbf{p}||}{2} = \frac{\sqrt{3}}{2} \neq 1$$

The former is linear in the coordinates x, y; the latter is linear in the distance $||\mathbf{p}||$. [15%]

(iii) Adding a first order polynomial would lead to a new RBF:

$$s(\mathbf{p}) = c_0 + c_1 x + c_2 y + \sum_{i=1}^{3} \lambda_i \phi(||\mathbf{p} - \mathbf{p}_i||)$$

The existence of this polynomial means we need to add *side conditions* which state that the sum of any first order polynomial, evaluated at the interpolation nodes and weighted with λ_i, must be zero. Hence the new matrix equation, for this specific case, is:

$$\begin{bmatrix} 0 & 2 & 2 & 1 & 0 & 0 \\ 2 & 0 & 2 & 1 & \sqrt{3} & 1 \\ 2 & 2 & 0 & 1 & \sqrt{3} & -1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & \sqrt{3} & \sqrt{3} & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The top-right 3×3 matrix comes from the new polynomial evaluated at each interpolation node. The bottom-left 3×3 matrix and the three lower zeros on the RHS are from the new side conditions. The bottom-right zero 3×3 matrix is because the actual polynomial coefficients c_0 to c_2 are not involved in these side conditions. [20%]

(iv) The initial data just represents one triangle, and the scalar data f_1 to f_3 at the vertices of this triangle can always be interpolated by a plane. Furthermore the plane would pass through zero at the origin, since f_1 (which is at the origin) is zero. This can be fully represented by the polynomial without any need for the basis functions $\phi(r)$. Hence there is a trivial solution (not required: you just need to note that the polynomial is all that is needed) with $\lambda_i = 0$, $c_0 = \frac{1}{\sqrt{3}}$ and $c_1 = c_2 = 0$. The RBF is hence:

$$s(\mathbf{p}) = \frac{x}{\sqrt{3}}$$

In that case, the RBF interpolant will produce exactly the same result (inside the triangle) as would a linear interpolation from a Delaunay triangulation. [15%]

Assessor's remarks: This was a question mainly on RBF interpolation. The calculations in (b)(i) were mostly answered very well, and most correctly spotted the difference between this and linear interpolation in (ii). Fewer candidates were able to correctly write out the required matrix in (iii), though there were several who noted nevertheless that this would have to create a different RBF and some even correctly noted the actual solution for (iv) where the RBF is entirely represented by the polynomial. Many students lost marks in (a) due to failing to answer all that was asked, or giving very imprecise answers.

4. **Gouraud and Phong shading, shaders**

(a) Both Gouraud and Phong shading work with *vertex normals*, which are found by averaging the normals of all polygons incident at a vertex. Gouraud shading proceeds by calculating an intensity at each vertex using the vertex normal and the Phong model. Intensities for interior pixels are found by bilinear interpolation. For efficiency, the interpolation can be formulated using fast, incremental calculations.

Phong shading interpolates the normals instead of the intensities. This tends to restore the original curvature of a surface, so that specular highlights can be reproduced accurately. The disadvantage of Phong shading is its expense. Even though the normals can be interpolated using incremental calculations, the interpolation considers the three components independently, so the vector must be renormalized at each pixel. Then, a *separate* intensity for each pixel is calculated using the Phong model.

Gouraud shading is comparatively fast, though it produces less photo-realistic renderings. It is particularly poor with the specular component. If a highlight should impinge on a polygon but not extend to its vertices, Gouraud shading will miss the highlight. [30%]

(b) Vertex and fragment shaders are programmable stages found in all modern Graphics Processing Units (GPUs). A vertex shader takes input vertices and transforms them into output vertices under the control of a user-defined program. This might involve coordinate transformations only (Phong shading) or lighting as well (Gouraud shading). The vertex shader's outputs are interpolated by a rasterizer to produce pixel data that is input to the fragment shader[1]. The fragment shader takes this interpolated data and runs another user-defined program to produce pixel outputs. This might involve very little (Gouraud shading) or full lighting calculations (Phong shading). Textures are also mapped at this stage. [20%]

(c) From the code and the variable names, the GLSL code must be calculating the $\mathbf{R}.\mathbf{V}$ part of the specular term in the Phong model, i.e.

$$I_s = \begin{cases} I_p k_s \cos^n \alpha = I_p k_s \left(\mathbf{R}.\mathbf{V}\right)^n & \text{for } \alpha, \theta < 90^\circ \\ 0 & \text{otherwise} \end{cases}$$

where the variable `shininess` contains the specular exponent n and `R1` is found by reflecting the illumination vector `nL1` in the surface normal `N`. From the `1` in the variable names, it would appear that there is more than one light source. For Gouraud shading, this code would appear in the vertex shader. [20%]

(d) Given the variable names and the fact that the code appears in a Phong vertex shader, where only coordinate transformations happen, this line of code would appear to apply the modelview matrix to a vertex normal, to transform the normal from local coordinates to view coordinates. The subsequent lighting calculations are presumably then done in view coordinates. However, we must take care to *rotate* the normals into view coordinates but not *translate* them too: the translation applies only to vertex coordinates, not normals! So

[1] In addition to rasterization, other operations that happen between the shaders include clipping, the perspective divide, mapping to 2D device coordinates and primitive assembly.

the `mat3()` function presumably picks out the upper 3×3 part of the 4×4 modelview matrix. `N` would need normalizing if the supplied variable `vertNormal` were not already normalized and/or the modelview matrix included any sort of scaling. [20%]

(e) The vertex normals undergo linear interpolation in the rasterizer between the vertex and fragment shaders. Element-by-element interpolation between two unit vectors does not generally output a unit vector: hence the need to renormalize `N` in the fragment shader. [10%]

Assessor's remarks: This question tested candidates' understanding of shading and shaders. It was generally well answered, with the vast majority of candidates demonstrating good knowledge of Gouraud and Phong shading, and coming up with good justifications for the need to renormalize normals in Phong vertex and fragment shaders. Not many candidates deduced the purpose of the GLSL `mat3()` function. More surprisingly, not many candidates could explain what a shader actually is, and what happens to data between the vertex and fragment shaders. This suggests a failure to engage with the more practical parts of the course, specifically the examples paper question that asked students to play with shaders in supplied code, and the lab.

Andrew Gee & Graham Treece
May 2026