Engineering Tripos Part IIA THIRD YEAR

Module 3G4: Medical Imaging & 3D Computer Graphics

Solutions to 2011 Tripos Paper

1. **Fundamentals of computed tomography**

(a)

$$\begin{aligned} a+c &= 18 \\ b+d &= 24 \\ b+c &= 30 \\ a+b &= 26 \end{aligned}$$

[10%]

(b)

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 18 \\ 24 \\ 30 \\ 26 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} = M$$

[10%]

(c) The most obvious ways to proceed are either to solve the equations by substitution, or to invert the 4×4 matrix M. By substitution:

$$\begin{aligned} a &= 18 - c \\ b &= 26 - a = 8 + c \\ c &= 30 - b = 22 - c \\ \Rightarrow c &= 11 \\ \text{so } b &= 19 \\ a &= 7 \\ \text{and } d &= 24 - b = 5 \end{aligned}$$

So $a = 7$, $b = 19$, $c = 11$ and $d = 5$. [20%]

(d) There are four key differences: scale, redundancy, consistency and noise.

The first of these is obvious. In a typical CT scan there is a much finer resolution in the computed image and vastly more projection data. Suppose that the image

is 256×256, it would not be efficient to invert a $256^2 \times 256^2$ matrix by Gaussian elimination.

In medical image data there will never be the same number of equations as unknowns. The projection data will contain a substantial element of redundancy and the solution method must therefore take all this richness into account.

Redundancy brings with it the problem of inconsistency. It is unlikely that a solution will exist that exactly satisfies all of the constraints implicit in the projection data. The solution will need to include a method of balancing these constraints and producing the best compromise solution. The easiest way to do this is to calculate the solution that results in the smallest mean squared error between predicted and measured projection data.

Finally, the projection data will include systematic measurement errors and random noise. Examples include cupping in CT data, statistical variations in the gamma photon reception in SPECT data, and simple measurement noise. In SPECT data, where there is a high level of noise, it is necessary to use algorithms that can model the noise and compensate for it, such as maximum likelihood expectation maximisation (ML-EM). [40%]

(e) In this case, the matrix M is given by

$$M = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

This is singular, so there is no unique solution. Any combination of values that satisfies $a = 3 - b = 4 - c = d - 3$ is a possible solution. It is therefore not possible to solve for the linear attenuation coefficients of the rods. [20%]

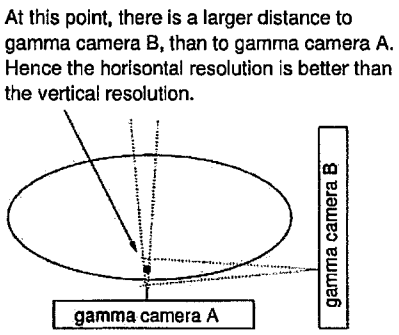
2. **Medical imaging modalities and resolution**

(a) (i) In two-dimensional ultrasonic imaging, the axial resolution is mainly limited by the pulse centre frequency. The lateral resolution is determined by the geometry of the transducer array and the combination of apodisation and delays that comprise the beam-forming algorithm. It also depends on whether static or dynamic receive beam-forming is employed. The elevational (out of plane) resolution is determined by the geometry of the acoustic lens used to focus the beam in the elevational direction.

Resolution in the lateral and elevational directions vary significantly with the depth in the image. In the elevational direction the resolution is highest at the focus of the acoustic lens. The same is true in the lateral direction, where the best resolution is at the transmit-focus of the beam-former (assuming dynamic receive beam-forming). The lateral resolution also degrades at the extreme edges of the B-scan image where a symmetrical excitation pattern is not possible. The axial resolution degrades slightly with depth due to pulse dispersion and poor signal-to-noise ratio. [25%]

(ii) In X-ray computed tomography, the in-plane resolution is predominantly determined by the beam width, the geometry and granularity of the receiver array and the number of projection images that are recorded. The out-of-plane resolution is determined by the beam width and the distance between the slices. CT resolution is isotropic within each scan-plane, but different, and usually lower perpendicular to the scan-plane. The resolution does not vary greatly as a function of the location within each slice. [25%]

(iii) In SPECT imaging, the resolution is determined by the geometry of the collimator, scintillation crystals and detectors. It also depends on the amplitude of the emitted radiation being appropriate. If there is too much radiation, there will be clashes between received gamma photons which will be removed by energy filtering. If there is too little radiation, the signal-to-noise ratio will be poor. In either of these cases it will be difficult to form an image. The resolution will vary with location in the data because of the spatial variability of the "collimator resolution." The effect of this is to give lower resolution at points in the middle of the subject that are not close to the camera at any point during the scan. This effect will also create anisotropy because a source will be located most accurately in directions perpendicular to the face of a nearby collimator. See the diagram below.



[25%]

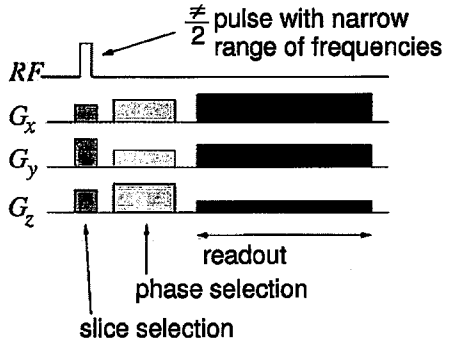
(b) Spatial encoding in magnetic resonance imaging is achieved using three different configurations of the gradient fields to modulate the response in three orthogonal directions. In relation to a spin-echo sequence, we have the following three stages.

During the first gradient field configuration, the RF pulse is applied. This results in *slice selection:* as only the slice in the field with a resonant frequency the same as the RF pulse gets rotated by $\pi/2$.

The second gradient field configuration is applied for a short time and serves to apply a spatially varying phase encoding to the response. This is achieved because different strengths of field leave the spins with different phase values. This is called *phase selection.*

The final gradient field configuration is applied during the *readout* phase while the MR data is recorded. This results in spatially-varying frequency modulation of the signal.

The overall result of these three constraints is to enable the MR response of each point in 3D space to be individually identified.



[25%]

3. *Mesh manipulation and contour areas*

(a) (i) For A, the first step is to find which vertex the user has clicked on, which can only be done by searching through the list of triangles working out the distance to the click point and selecting the triangle vertex with the minimum distance. However, it is likely that several triangles will contain this vertex, and the floating point numbers may not be identical, so some sort of distance margin may be necessary to locate them. Once the user has moved the vertex, the new location needs to be recorded in all of these triangles.

For B, the first step involves searching the point list and finding the closest point to where the user clicked. The coordinates of this point can be adjusted once the user has moved the vertex: the triangle list remains unchanged. Clearly this is better than option A, since there are three times fewer points to search initially, only one to update, and no ambiguity.

(ii) For A, having found the points in all relevant triangles as in (i), these triangles can be removed from the list. No other updating is necessary. For B, having found the point and deleted it, the deletion will affect the indices of other points in the list, and hence all the vertices in the triangle list have to be checked and possibly updated to reflect the changed indices, as well as removing any triangles which actually contain the vertex.

In this case option A is probably the easiest option, since it does not require re-indexing of the triangle list. The time advantage in searching through fewer vertices to find the selected vertex in option B is almost certainly lost in the subsequent re-indexing of the triangle list.

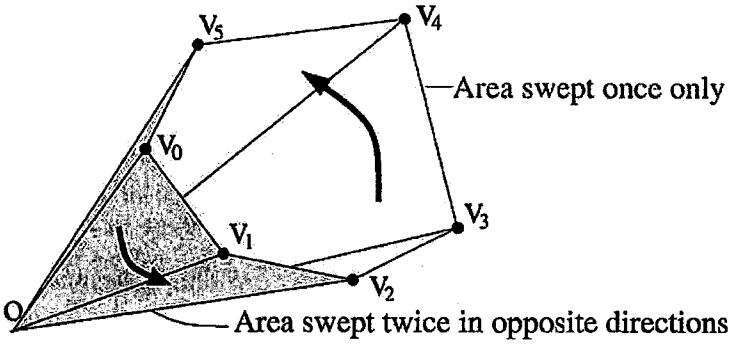
(iii) Finding an edge is a more complex operation. In this case we need to check each side of each triangle to see whether the click point lies on a line joining the two vertices either side of the edge. This is most easily done by looking at the scalar product of a normalised vector from the click point to each of two vertices: if this is -1, then the click point lies on the edge in question.

For option A, we need to perform this check three times for each triangle. Having found the triangles containing the selected edge, these should be deleted (since they will have zero area after the edge is reduced). However, we also need to replace *both* vertices in any other triangles containing them with a new vertex positioned somewhere along the original edge. So this necessitates a further search through the triangle list.

For option B, the operation is nearly the same — we still need to search through the *triangle* list to find out which edge the user clicked on, since this tells us which points are joined together at the edges. However, we will need to refer to the point list to do the calculations. Having found and deleted the triangles containing the edges, we now have to replace *one* of the vertices (say v_1) in the point list with a new vertex, and delete the other one (say v_2). Then the triangle list must be searched again, both to re-label any reference to v_2 with v_1 and also to re-index the vertices since we have deleted one, as in (ii).

There is not much to choose between representations for this operation: we really need to use an edge-list representation to make this task easier.

(b) (i) There are several methods for finding the area of a closed contour. Probably the easiest is to take the sum of the areas of all the triangles formed from joining each line in the contour to the origin:



$$A = \frac{1}{2}\left\{(\mathbf{v_N} \times \mathbf{v_1}) + \sum_{1}^{N-1}(\mathbf{v_n} \times \mathbf{v_{n+1}})\right\}$$

(ii) The B-spline geometry matrices each consist of four contour points, and in order to maintain continuity, each segment must only introduce one new point. The segment is drawn between the second and third points in the geometry matrix, so the matrices, in order of segment number, are:

$$\begin{bmatrix} x_3 & y_3 \\ x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}, \quad \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}, \quad \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_0 & y_0 \end{bmatrix}, \quad \begin{bmatrix} x_2 & y_2 \\ x_3 & y_3 \\ x_0 & y_0 \\ x_1 & y_1 \end{bmatrix}$$

This representation will only approximate the contour vertices, not interpolate them, so it is only a good idea if the vertices are reasonably dense, or if they contain noise (for example hand tremor) which it is desired to remove.

(iii) We can calculate the enclosed area of the B-spline representation by breaking each segment down into small parts and applying the equation above for calculating the polygonal area. Taking a *single* segment:

$$\begin{aligned} A_n &= \frac{1}{2}\sum_t \mathbf{v_n}(t) \times \mathbf{v_n}(t+\Delta t) \\ &= \frac{1}{2}\sum_t \mathbf{v_n}(t) \times (\mathbf{v_n}(t) + \mathbf{v_n}'(t)\Delta t) \\ &= \frac{1}{2}\sum_t \mathbf{v_n}(t) \times \mathbf{v_n}'(t)\Delta t \end{aligned}$$

In the limit, this becomes the integral:

$$A_n = \frac{1}{2}\int_0^1 \mathbf{v_n}(t) \times \mathbf{v_n}'(t)\ \mathrm{d}t$$

which can be evaluated numerically for each B-spline segment.

4. *Radial basis function interpolation*

(a) (i) $\phi(r_n)$ is the basis function. There are as many of these as there are measurements to be interpolated, and each is a function of r_n, the distance between the current location and the measurement location. Since $s(x, y)$ is made up of a sum of these functions, the smoothness and continuity of $s(x, y)$ is largely based on the basis functions ϕ.

The polynomial $p(x, y)$ exists to ensure that the system of equations is solvable: it may or may not be needed, depending on the type of basis function chosen and the domain over which measurements are made. $s(x, y)$ is increasingly dominated by this polynomial function far away from the measurement locations.

The constants λ_n weight each basis function, and are calculated to ensure that $s(x, y)$ interpolates a given set of data.

(ii) The matrix equation is based on *interpolation* and *side* conditions. The interpolation conditions, in the upper rows of the matrix equation, state that if $s(x, y)$ is evaluated at one of the measurement locations, the original measurement should be returned: i.e. it is an interpolating function. Hence there is one row per measurement, each evaluating $s(x_n, y_n) = f_n$ for $1 \leq n \leq N$.

The remaining three rows represent the side conditions. These state that the weighted sum of any polynomial of the same degree as $p(x, y)$, evaluated at the measurement

locations and weighted by the λ_n values, should equal zero:

$$\sum_{n=1}^{N} \lambda_n q(x_n, y_n) = 0$$

Since this has to be true for any polynomial $q(x, y)$, this can be interpreted as:

$$\sum_{n=1}^{N} \lambda_n y_n = \sum_{n=1}^{N} \lambda_n x_n = \sum_{n=1}^{N} \lambda_n = 0$$

The purpose of these conditions is to ensure that the matrix system has a unique solution and hence is solvable.

(b) (i) The matrix in this case is:

$$\begin{bmatrix} 0 & 1 & 1 & \sqrt{2} & 1 & 0 & 0 \\ 1 & 0 & \sqrt{2} & 1 & 1 & 1 & 0 \\ 1 & \sqrt{2} & 0 & 1 & 1 & 0 & 1 \\ \sqrt{2} & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

(ii) The lowest row gives:

$$\lambda_3 = -\lambda_4$$

the next row up gives:

$$\lambda_2 = -\lambda_4$$

and combined with the third from last row, we have:

$$\lambda_1 = \lambda_4$$

Using these substitutions and considering the first row:

$$c_0 = (2 - \sqrt{2})\lambda_4$$

and from the second row:

$$(2 - \sqrt{2})\lambda_4 + c_0 + c_1 = 1$$

Hence:

$$c_1 = 1 - 2(2 - \sqrt{2})\lambda_4$$

And similarly for the third row:

$$c_2 = 1 - 2(2 - \sqrt{2})\lambda_4 = c_1$$

Finally, the fourth row gives:

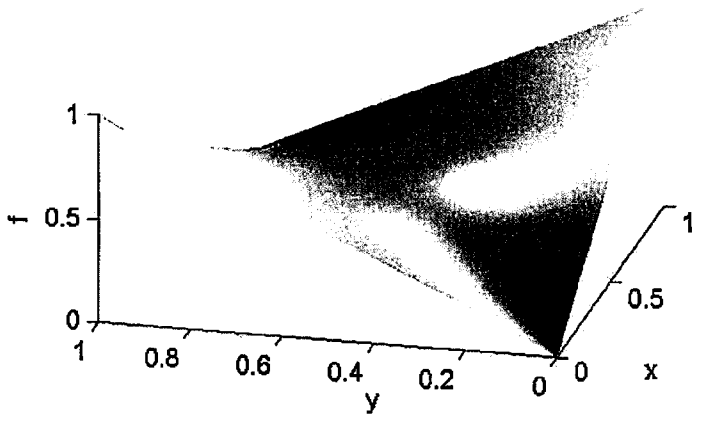
$$(\sqrt{2}-2)\lambda_4 + c_0 + c_1 + c_2 = 0$$

Hence

$$\lambda_4 = \frac{1}{2(2-\sqrt{2})}$$

and finally:

$$\begin{aligned}
\lambda_1 &= \frac{1}{2(2-\sqrt{2})} \\
\lambda_2 &= -\frac{1}{2(2-\sqrt{2})} \\
\lambda_3 &= -\frac{1}{2(2-\sqrt{2})} \\
\lambda_4 &= \frac{1}{2(2-\sqrt{2})} \\
c_0 &= \frac{1}{2} \\
c_1 &= 0 \\
c_2 &= 0
\end{aligned}$$



(iii) $s(x, y)$ is equal to 0.5 at all these locations. In fact it is 0.5 everywhere on the lines $x = 0.5$ and $y = 0.5$, which can also be seen from the symmetry of the RBF and the measurement locations and data. For reference (not required in the question), the full function is shown above.

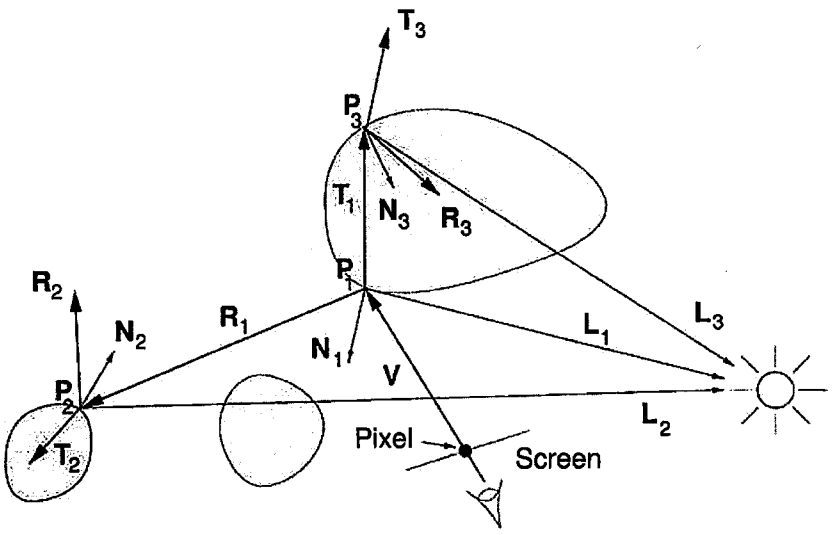
(c) There are two possible Delaunay triangulations of the measurement locations, one with a diagonal connection between $(0, 0)$ and $(1, 1)$ and one with a diagonal

connection between $(0,1)$ and $(1,0)$. In the former case, linear interpolation would return the value 0 for the mid-point $(0.5, 0.5)$, in the latter case it would return 1 at this point. Hence the interpolated data is highly dependent on which triangulation is chosen, and unlike the RBF neither triangulation results in an interpolant which has the same symmetry as the original data.

The interpolated values along the sides of the cube $(0,0)$ to $(0,1)$ etc. are very similar for the Delaunay triangulation and the radial basis function, though the radial basis function does not give an exactly linear variation between 0 and 1: $s(x,y)$ returns 0.26 at $(0, 0.25)$.

5. **Ray tracing and environment mapping**

(a) Recursive ray tracing is a rendering technique which accounts for a degree of indirect illumination. It elegantly combines hidden surface removal, transparency effects and shadow computation into a single model.



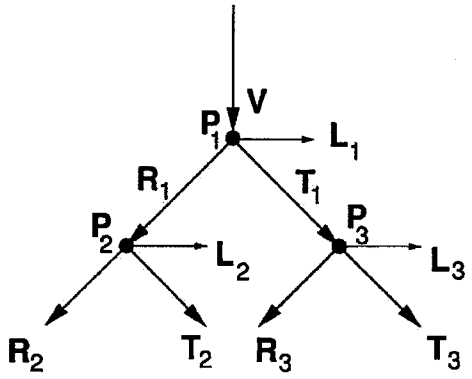
Simple ray tracing algorithms operate in world coordinates. A ray $\mathbf{V}$ is fired from the centre of projection through every pixel on the screen. When the ray strikes an object at $\mathbf{P}_1$, further rays are spawned. One ray, $\mathbf{L}_1$, heads for the ith light source. If $\mathbf{L}_1$ passes through other objects on the way, then the illumination intensity is attenuated by a shadow factor S_i, depending on the number and opacity of objects in the way.

As well as these **shadow feelers**, the algorithm also spawns a **reflection ray** $\mathbf{R}_1$ and a **refraction ray** $\mathbf{T}_1$. The direction of the refraction ray is determined from the refraction indices using Snell's law. The intensity of the pixel is then

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

where $I_{r\lambda}$ is the intensity of the reflection ray and $I_{t\lambda}$ is the intensity of the refraction ray. k_t is a **transmission coefficient** in the range 0 to 1.

Values for $I_{r\lambda}$ and $I_{t\lambda}$ are found by recursively evaluating the equation at the surfaces $\mathbf{R}_1$ and $\mathbf{T}_1$ next intersect ($\mathbf{P}_2$ and $\mathbf{P}_3$), with $\mathbf{P}_1$ as the new viewpoint. So the progress of the algorithm follows a **ray tree**:



The tree is *constructed* top down, until either a ray fails to intersect an object or some predetermined maximum depth is reached. The tree is then *evaluated* bottom-up, as each node's intensity is computed from its children's intensities.

Compared with standard surface rendering, recursive ray tracing produces more photo-realistic results but requires much more computation. However, the ray tracing algorithm is easily parallelized and could run at a reasonable (but not interactive) rate on a multiprocessor machine, especially if bounding boxes are used to speed up intersection tests. [50%]

(b) No, back-face culling and ray tracing do not mix. Even if the scene comprises only solid (not hollow) polyedhra, polygons oriented away from the camera may well be visible reflected in other surfaces, so cannot be ignored. [10%]

(c) The obvious advantage of environment mapping is speed. Everything fits into the standard surface rendering pipeline and can therefore take advantage of standard graphics hardware. Even the calculation of the mirror rays R is similar to what has to be done for specular lighting calculations. OpenGL and DirectX both support cubic environment mapping, and it runs at interactive rates on modest graphics cards.

While environment mapping certainly gives the impression of the environment reflected in the shiny object, ray tracing provides far more accurate rendering. Here are some shortcomings of environment mapping.

- There is no parallax in the reflection, it is as if everything were a very long way away. In reality, if the reflection includes, say, a nearby tree in front of some distant mountains then, as the object moves, the reflected tree should also move

with respect to the reflected mountains. But this clearly cannot happen with environment mapping, since the texture maps are static and capture just one view of the tree and the mountains.

- Only the environment is reflected, not any inter-reflections from the object itself. This is only accurate if the object is convex.
- Refraction is not handled at all.
- The reflection will not respond faithfully to lighting changes.

There is also the matter of acquiring and filtering the six texture maps with appropriate care. Look carefully at the figure in the question and observe how there are no discontinuities at the seams between the cube's faces. This is critical to avoid obvious artefacts in the reflection. [40%]

6. **Interpolation and texture mapping**

(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 a colour at each vertex using the vertex normal and the Phong model. Colours for interior pixels are found by bilinear interpolation. For efficiency, the interpolation can be formulated using fast, incremental calculations.

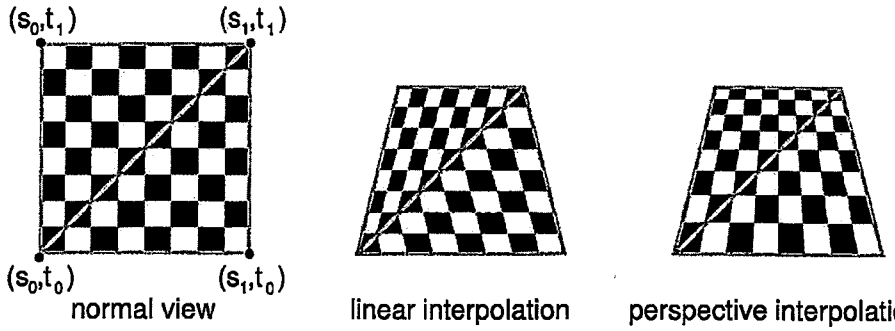
Phong shading interpolates the normals instead of the intensities. Even though the normals can be interpolated using incremental calculations, the interpolation handles the three components independently, so the vector must be renormalized at each pixel. Then, a *separate* intensity for each pixel is calculated using the Phong model.

For both shading methods, bilinear interpolation is used to derive depth (z_s) values within the interior from those at the vertices. [25%]

(b) (i) C is half way along the rendered edge from A to B, so $\alpha = 0.5$. Using the linear equation, we obtain $t_{\alpha=0.5} = \frac{1}{2}t_0 + \frac{1}{2}t_1$. The more complicated equation gives

$$t_{\alpha=0.5} = \frac{0.5(t_0/10) + 0.5(t_1/20)}{(0.5/10) + (0.5/20)} = \frac{2t_0 + t_1}{2+1} = \frac{2}{3}t_0 + \frac{1}{3}t_1$$

So the linear equation maps the texture point $(s_0, \frac{1}{2}t_0 + \frac{1}{2}t_1)$ to C, i.e. half way up the left edge of the map. This is clearly wrong, since C is not half way along the chessboard in object coordinates: considering the perspective compression of distant objects, C must be nearer A than B. The more complicated equation maps the texture point $(s_0, \frac{2}{3}t_0 + \frac{1}{3}t_1)$ to C, i.e. a third of the way up the left edge of the map. This is more plausible: in fact, it is *perspective correct* interpolation, though a formal proof is not required here. The figure below shows the likely appearance of the renderings.



[25%]

(ii) Linear interpolation should be used for z_s values, so C's z_s value would be half way between those of A and B. Note how this would place C's z_v value nearer to A's than B's, which is correct. The perspective mapping from view coordinates (z_v) to 3D screen coordinates (z_s) already contains the reciprocal of z_v that is required for perspective correct interpolation. Consequently, simple linear interpolation can be used for the z_s values. [20%]

(c) If the chessboard maps to a small number of screen pixels, we need to be very careful when sampling the texture map. Suppose, say, there are just ten pixels between A and B. The perspective correct interpolation will furnish ten (s_0, t_α) texture coordinates, but if we just use these to sample the ten nearest pixels from the texture map, we can expect a fairly random outcome, depending on whether those ten coordinates happened to fall on black or white pixels. This is essentially an aliasing artefact caused by severe undersampling of the texture map.

Better would be to generate a smaller version of the texture map, of about the same size as the rendered chessboard, by proper downsampling with anti-aliasing smoothing. Then we could safely index into this smaller texture map, perhaps using linear interpolation between the texture pixels neighbouring each (s_0, t_α) value.

To avoid the computational expense of having to calculate the downsampled texture map on the fly, it is common to generate a set of suitable texture maps in advance, for instance of sizes 256×256, 128×128 ... 4×4, 2×2 and 1×1. This set of images is known as a mipmap. Then, at render time, the graphics engine selects the most suitable map according to the rendered area, or perhaps even interpolates between the two most suitable maps.

For our chessboard, the smaller maps, such as the 2×2 one, would just be a uniform grey: this is what you get if you repeatedly smooth and downsample the chequerboard image. A small, distant chessboard rendered with the 2×2 map would appear uniformly grey. This is definitely the right thing to do though! Nearest neighbour sampling of the original, high resolution map would instead furnish a random collection of black and white pixels, which would change with viewing angle, producing a highly distracting artefact in animated sequences. [30%]

Andrew Gee, Richard Prager & Graham Treece, December 2010